Curso De Clipper 5.2

TOTOLAN, MICHOACAN

M E X I C O

DICIEMBRE, 1997.

Impartido Por:

L.I. Martín Alejandro Guerra Hernández

UNIDAD I INTRODUCCION A CLIPPER

1.1. Origen de clipper.

Clipper nació como compilador de dbase. Permitía que las instrucciones de dbase se ejecutarán más rápido. Sus inicios fueron en el año de 1987 con la versión Clipper Summer 87 sus creadores Brian Rusell y Rich McConell, con los años ha aumentado el conjunto de instrucciones de dbase para incluir potentes ordenes que solo el compilador de Clipper reconoce.

1.2. Características de clipper.

Es un sistema de desarrollo de aplicaciones de base de datos basado en un lenguaje de alto nivel. Permite la creación y explotación de sistemas profesionales de computadoras y redes locales.

Se compone de un preprocesador, un compilador, un enlazador, un depurador interactivo, un editor de programas, un generador de reportes y un manejador de base de datos.

Fue diseñado para fomentar la programación modular.

1.3. Uso comercial de clipper.

Nantucket comercializa Clipper, constituyendo un nuevo enfoque para la construcción de software, ofreciendo técnicas potentes que permitan producir aplicaciones flexibles y fáciles de mantener.

Es estos años a ganado más de 200,000 usuarios, haciendo más productivas y fáciles sus tareas (en el año de 1992).

UNIDAD II CONOCIMIENTOS BASICOS

2.1. Tipos de datos.

2.1.1. Arreglos o matrices.

El arreglo es una estructura compuesta por varios componentes almacenados consecutivamente en memoria.

Una matriz es un conjunto de valores relacionados entre sí que comparten un mismo nombre. Cada valor se reconoce como elemento. La matrices o arreglos se crean de acuerdo a:

Sintaxis: DECLARE <identificador> [tamaño]
<identificador> = ARRAY(tamaño)
[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> [tamaño]

Ejemplo: declare un arreglo de tamaño 10, posteriormente una matriz de 5 por 3 de tamaño.
DECLARE calif[10] o calif = ARRAY(10)
LOCAL materia[5,3] o materia =ARRAY(5,3)

2.1.2. Carácter.

El tipo de carácter se emplea para datos que se desea manejar como cadenas de caracteres de longitud fija. Para expresar una cadena sin valor emplee un par de delimitadores sin caracteres intermedios, puede ser el apóstrofe (´) o comillas(”). Para declarar este tipo de dato siga:

Sintaxis: STORE <”valor”> To <identificador>
<identificador> = <”valor”>
STORE SPACE(tamaño) To <identificador>
[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> := <”valor”>
[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> := SPACE (tamaño)

Ejemplo: declare una variable con un valor de S, posteriormente una variable de tamaño 19.
STORE “S” To resp o resp = ”S”
LOCAL nom:= SPACE(19) o STORE SPACE(19) To nom

2.1.3. Numérico.

El tipo numérico se emplea para los datos que desea manejar matemáticamente. El conjunto de caracteres va del dígito 0 al 9. Los números van desde 10^{-308} hasta 10^{+308} . Su precisión esta garantizada hasta 16 dígitos. Para declarar este tipo de dato siga:

Sintaxis: STORE <valor> To <identificador>
<identificador> = <valor>
[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> := <valor>

Ejemplo: declare una variable con un valor de 0.
STORE 0 To calif o calif = 0 o LOCAL calif := 0

2.1.4. Fecha.

El tipo de fecha se emplea para representar fechas. El conjunto de caracteres son dígitos del 0 al 9 y un separador. Las fechas van desde el 01/01/0100 a 12/31/2999. Para declarar este tipo de dato siga:

Sintaxis: <identificador> = CTOD ("MM/DD/AA")

[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> := CTOD ("MM/DD/AA")

Ejemplo: declare una variable con la fecha 12 de enero de 1999.

fech= CTOD("01/12/99") o LOCAL fech := CTOD("01/12/99")

2.1.5. Lógico.

El tipo lógico se utiliza para representar datos booleanos. Estos datos toman 2 valores, cierto o falso. El conjunto de caracteres de este tipo se compone de las letras y, Y, t, T (representan el valor **Verdadero**) n, N, f, F (representan el valor **Falso**). Para formar un valor lógico, se encierra el valor entre puntos. Los valores Para declarar este tipo de dato siga:

Sintaxis: STORE <.valor.> To <identificador>

<identificador> = <.valor.>

[PRIVATE | PUBLIC | LOCAL | STATIC] <identificador> := <.valor.>

Ejemplo: declare una variable con un valor de verdadero, posteriormente una variable de valor falso.

STORE .T. To resp1 o resp1 = .T. o LOCAL resp1 := .T.

STORE .F. To resp2 o resp2 = .F. o LOCAL resp2 := .F.

2.1.6. Memo.

El tipo memo se emplea para datos que se desea manejar como cadenas de caracteres de longitud variable. Solo puede existir como campo de una base de datos.

Se manejan como cadenas de caracteres, incluso tiene la misma limitación de 65,535 caracteres como longitud máxima. Dado que se trata de un campo de base de datos no tiene representación literal.

2.1.7. Nil.

El tipo nil se utiliza para manejar variables no inicializadas sin que se genere un error de ejecución. Tiene solo un valor, nulo o nil.

2.2. Operadores de clipper.

Un operador es un símbolo que identifica una operación básica.

Operadores matemáticos:

Símbolo	Operación
** o ^	Exponenciación
*	Multiplicación
/	División
%	Modulo o residuo
+,-	Suma, Resta

Operadores relacionales:

Símbolo	Operación
<	Menor que
>	Mayor que
=	Igual que
==	Exactamente igual
<>, #, !=	Diferente
<=	Menor o igual que
>=	Mayor o igual que

Operadores lógico:

Símbolo	Operación
.AND.	"Y" Lógico
.OR.	"O" Lógico
.NOT. o !	"NO" Lógico

Operadores de asignación:

Símbolo	Operación
=	Asignar
:=	Asignar en línea

2.3. Clases o tipos de variables.

Una variable puede tener uno de los cuatro tipos de visibilidad:

- *Una variable PUBLIC,* es visible para cualquier programa del sistema. En ocasiones se les denomina a estas, variables globales.
- *Una variable LOCAL,* solo se puede ver desde el procedimiento o función en el que se ha declarado. Cuando el control vuelve al programa principal o encuentra un RETURN, el valor de la variable se pierde.
- *Una variable STATIC,* es una variable especial. Debe ser declarada inmediatamente después de la sentencia PROCEDURE o FUNCTION. Una variable de este tipo solo es visible en el procedimiento o función donde se declara; sin embargo mantiene su contenido, de forma que sigue disponible la siguiente vez que se llama al procedimiento o función en donde se encuentra.
- *Una variable PRIVATE,* es visible desde todos los programas a los que llama la función o procedimiento en el que se ha declarado la variable. Si una variable no se declara específicamente se convierte en variable de tipo *Private*.

2.4. Palabras o funciones reservadas.

A continuación se mencionan las palabras reservadas usadas en Clipper:

Else	Enddo	If
Elseif	Endif	Procedure
Endcase	Function	While

Nota: Las palabras reservadas no pueden usarse como nombre de variables, procedimientos o funciones.

Algunas funciones reservadas de Clipper:

Bof	Eof	Rtrim
Chr	File	Select
Ctod	Found	Space
Col	Inkey	Setpos
Date	Lastrec	Str
Delete	Lower	Time
Dtoc	Ltrim	Upper

La diferencia entre una función reservada y una palabra reservada, radica en que las funciones tienen instrucciones ya definidas, siendo parte integral del compilador y una palabra nos permite desarrollar o crear funciones.

UNIDAD III CREACIÓN DE UN PROGRAMA

3.1. Recomendaciones al crear un programa.

Es necesario desarrollar la habilidad de programación de una forma que facilite el mantenimiento del código, tener consistencia y modularidad.

La computadora debe automatizar las tareas que consumen más tiempo y proporcionar informes oportunos para la toma de decisiones.

Para lograr realizar un sistema con óptimos rendimientos, debe tomar en cuenta los siguientes:

Programación modular, se define como el proceso de dividir grandes secciones de código en pequeñas secciones o módulos, que se puedan reutilizar una y otra vez.

Nomenclatura de un programa, otro aspecto importante es el asignar nombres a los archivos, campos, variables, procedimientos y funciones. El desarrollo de un buen método facilitara la lectura de los programas y evitara problemas.

Podría utilizar los dos primeros caracteres del nombre del archivo para describir el sistema general al que pertenece y el resto para el propósito del archivo. De igual forma, utilice nombres descriptivos para los campos de las bases de datos y para las variables de los programas.

Uso de mayúsculas y minúsculas, clipper no distingue entre mayúsculas y minúsculas. No obstante, le resultará más sencillo diferenciar las funciones y ordenes creadas por usted mismo si las escribe en minúsculas y las funciones de clipper en mayúsculas.

Sangrías dentro del programa, resulta más sencillo leer el código si se ha utilizado sangrías para separar algunas secciones de código. Las sangrías se utilizan cuando usa estructuras de control tales como: DO WHILE/ENDDO, IF/ENDIF, DO CASE/ENDCASE, FOR/NEXT, Funciones o Procedimientos. Al usar sangría se mejora la legibilidad de los programas creados por el usuario. La diferencia la aprecia en el siguiente ejemplo:

```
DO WHILE .T.
DO CASE
CASE estacion == verano
IF jardin
DO plantar
ENDIF
CASE estacion == otoño
DO cosecha
OTHERWISE
DO nada
ENDCASE
ENDDO
```

```
DO WHILE .T.
   DO CASE
     CASE estacion == verano
       IF jardin
          DO plantar
       ENDIF
     CASE estacion == otoño
       DO cosecha
     OTHERWISE
       DO nada
   ENDCASE
ENDDO
```

Documentación del código, el hecho de denominar las funciones y variables de la forma más adecuada posible, es uno de los pasos más importantes para documentar el código. En algunos casos es imposible explicar todo lo que se intenta hacer con un programa, por muy descriptivos que sean los nombres utilizados. Utilice el asterisco (*), doble barra inclinada (//), doble ampersand (&&) o barra inclinada - asterisco (/*) y asterisco - barra inclinada (*/).

Se debe seguir buenas practicas de documentación del código, par que el código sé más comprensible y se pueda identificar la lógica usada en la creación del código.

3.2. Uso de un editor.

El editor de programas de clipper es el PE. Con él puede crear y modificar sus programas. PE se localiza en C:\clipper5\bin, para iniciar su sesión de trabajo teclee:

Sintaxis: PE <nombre de archivo>

Nota: Si no se especifica el nombre del archivo, se crea un nuevo archivo en blanco.

Ejemplo: si desea modificar el programa llamado MENU.PRG, entonces teclee.

```
C:\clipper5>PE MENU
```

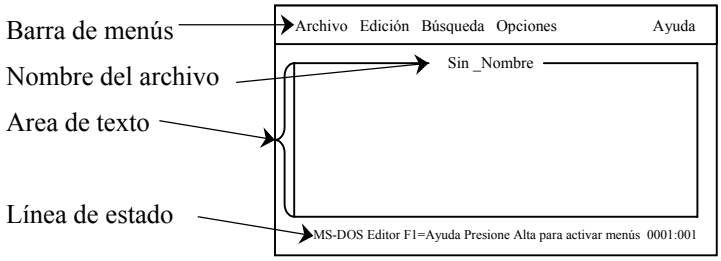
A continuación se describe el uso de las teclas más comunes del PE:

T E C L A	*A C C I O N*
↑ ↓	Sube, baja un renglón.
← →	Un carácter a la izquierda o derecha.
Inicio Fin	Inicio o Final de la línea.
Ctrl+AvPag Ctrl+RePag	Inicio o final del archivo.
AvPag RePag	Pantalla anterior o posterior.
↵	Inicio de un nuevo renglón.
Supr	Elimina el carácter donde esta el cursor.
←(Retroceso)	Elimina el carácter a la izquierda.
Ctrl+Y	Elimina un renglón.
Ctrl+W	Graba el archivo y sale del PE.
Alt+W	Graba el archivo y continua en el PE.
Alt+S	Búsqueda de una cadena.

Pero este editor es demasiado limitado, puede usar cualquier editor de su preferencia, pero recuerde grabar su programa con 8 caracteres como máximo y con la extensión PRG. En nuestro caso usaremos el editor del MS-DOS (EDIT.COM).

Conviene recordar que un editor de textos no es un procesador de textos, como por ejemplo: Word, WordPerfect. Un procesador de textos proporciona muchas características como distintos estilos de fuente, alineación, márgenes, longitud de pagina, etc.

Al usar el EDIT.COM, aparecerá al inicio:


Barra de menús
Nombre del archivo
Area de texto
Línea de estado
Archivo Edición Búsqueda Opciones Ayuda
Sin _Nombre
MS-DOS Editor F1=Ayuda Presione Alta para activar menús 0001:001

Barra de menús, presenta los menús y comandos que puede utilizar.
Nombre del archivo, contiene el nombre del programa actual.
Area de texto, sección de trabajo donde puede teclear su programa.
Línea de estado, presenta mensajes e información de los comandos activos en ese momento, así como él numero de Línea y de Columna.
Para activar los menús, haga lo siguiente:

M E N U	*TECLA DE ACTIVACION*
Archivo	Alt + A
Edición	Alt + E
Búsqueda	Alt + B
Opciones	Alt + O
Ayuda	Alt + U

Descripción de algunos aspectos importantes para el uso del Editor:

- En el área del texto teclee su programa, si desea *escribir en otra línea* pulse ↵.
- Para *desplazarse por el texto:*

T E C L A	*A C C I O N*
↑ ↓	Sube, baja un renglón.
← →	Un carácter a la izquierda o derecha.
Inicio Fin	Inicio o Final de la línea.
Ctrl+Inicio Ctrl+Fin	Inicio o final del archivo.

- Para *seleccionar texto:*

T E C L A	*A C C I O N*
Mayus + ← o →	Una palabra.
Mayus + Inicio o Fin	Una línea.
Mayus + ↑ o ↓	Un párrafo.

- Para *borrar texto:*

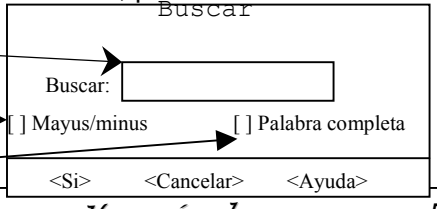
T E C L A	*A C C I O N*
Supr	Elimina el carácter donde esta el cursor.
←(Retroceso)	Elimina el carácter a la izquierda.
Ctrl+Y	Elimina una línea.
Párrafo	Selecciónelo y presione la tecla Supr.

- Para *copiar texto*, seleccione la línea o párrafo que desea copiar y del menú Edición elija Copiar.

- Para *pegar texto*, coloque el cursor donde desea colocar el texto y del menú Edición elija Pegar.

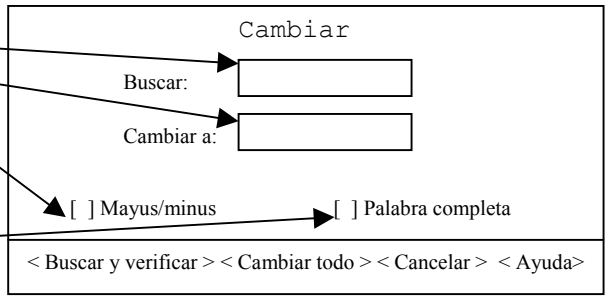
- Para *cortar texto*, seleccione la línea o párrafo que desea cortar y del menú Edición elija Cortar.

- Cuando desee encontrar un termino, una variable o un nombre de una función, para ello utilice del menú Búsqueda el comando Buscar, aparece el cuadro:



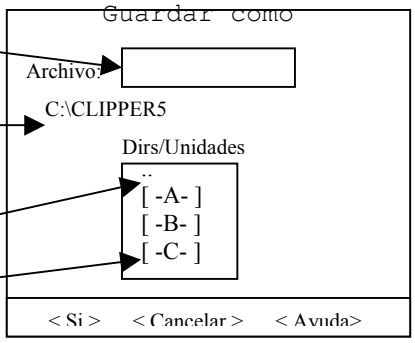
En el cuadro *Buscar*, escriba la palabra que desea buscar.
En la casilla de verificación Mayus/minus actívela, si desea que la palabra que introdujo se diferencie de otras palabras como mayúscula o minúscula.
En la casilla Palabra completa actívela, si desea que la palabra que introdujo tome en cuenta una palabra completa o no.

- En ocasiones decidirá cambiar un termino o una variable por otra, para ello utilice Cambiar del menú Búsqueda, aparece el cuadro:



En el cuadro *Buscar*, escriba la palabra que desea buscar.
En el cuadro *Cambiar a*, escriba la palabra por la que desea se cambie la palabra anterior.
En la casilla de verificación Mayus/minus actívela, si desea que la palabra que introdujo se diferencie de otras palabras como mayúscula o minúscula.
En la casilla Palabra completa actívela, si desea que la palabra que introdujo tome en cuenta una palabra completa o no.

Cuando termine de teclear su programa y desea *guardarlo por primera vez*, o con un nombre diferente al que tiene, del menú Archivo elija Guardar como, aparece el cuadro:

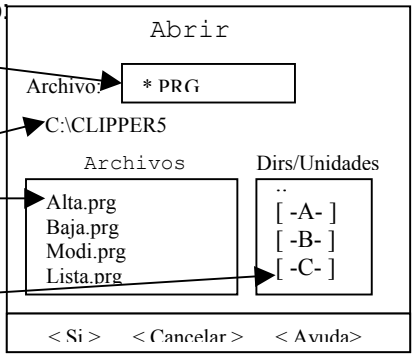


En el cuadro *Archivo*, escriba el nombre del archivo (recuerde no exceder de 8 caracteres como máximo y agregar la extensión.PRG).

En esta línea muestra la ruta (Unidad \ Directorio y/o Subdirectorio).

Muestra las unidades y/o subdirectorios disponibles en ese momento.
.. Indica un directorio y/o subdirectorio anterior.
[A] nos permite cambiar a la unidad A.
[C] nos permite cambiar a la unidad C.

Observe su editor, verá que en la parte superior de la pantalla aparece Sin Titulo y debe cambiar al nombre que usted eligió.
Si desea *abrir un archivo,* del menú Archivo elija Abrir, aparece el cuadro:



En el cuadro *Archivo*, escriba el nombre del archivo (en nuestro caso escriba *.PRG, para que muestre todos los archivos de programa disponibles).
En esta línea muestra la ruta (Unidad \ Directorio y/o Subdirectorio).

Muestra las unidades y/o subdirectorios disponibles en ese momento.
.. Indica un directorio y/o subdirectorio anterior.
[A] nos permite cambiar a la unidad A.
[C] nos permite cambiar a la unidad C.

Si desea cerrar el archivo que se encuentra activo, del menú Archivo elija Cerrar.
Para abandonar el editor de textos del menú Archivo elija el comando salir.
Cuando teclea o modifica un archivo(programa) y no guardo los cambios, aparecerá un cuadro de advertencia que le pregunta si desea guardar los cambios del archivo, la contestación depende de usted, si desea guardar cambios diga Si de lo contrario diga No.

3.2. Compilación y enlace de los programas.

Después de que termino de teclear su programa, debe verificar sino contiene errores, es decir debe Compilarlo. En la compilación Clipper evalúa cada una de las líneas del programa. El compilador comprueba si las ordenes y funciones se han introducido con la sintaxis correcta, en caso contrario genera mensajes de error. Existen dos maneras de compilar:

Sintaxis: CLIPPER <nombredelprograma> <lista de opciones>
CL <nombredelprograma>

nombredelprograma, debe especificar el nombre del programa a compilar. Nota: solo debe escribir el nombre del programa *sin la extensión PRG*.

Opciones: /A especifica que las variables declaradas con Private, Public sean variables de memoria
/M solo compila el programa actual (PRG) omite las referencias DO, SET PROCEDURE.
/V especifica que las variables no declaradas o sin alias se traten como Private o Public

El primer método compila uno o varios archivos de programa (PRG), sino se encuentran errores, genera el código objeto (OBJ). Este método ocupa un enlazador, que se encarga de combinar los archivos objeto (OBJ) y después generar el código ejecutable (EXE).

Sintaxis: RTLINK FI <nombredelprograma>

El segundo método compila uno o varios archivos de programa (PRG), sino se encuentran errores, genera el código objeto (OBJ) y automáticamente realiza el enlace para generar el código ejecutable (EXE).

Ejemplo: si desea compilar el programa llamado MENU.PRG, entonces teclee.

C:\clipper5>CL MENU ↵

3.4. Ejecución de un programa.

Después de realizar la compilación y el enlace de los programas, y a través de estas acciones ver generado el código objeto y ejecutable, ahora debe ejecutar su programa.

Busque en la ruta donde compilo su programa (PRG), el nombre del programa con la extensión (EXE), si lo encuentra, esto nos indica que ya genero el código ejecutable. Ahora escriba el nombre del programa ejecutable y pulse ↵.

Por medio de la ejecución del programa, usted prueba si su programa es *funcional* (realiza lo que el programador quiere), *eficiente* (que la ejecución y las búsquedas sean rápidas) y *transportable* (poder ejecutarse en cualquier computadora).

Ejemplo: si desea ejecutar el programa llamado MENU.EXE (previamente compilado), entonces teclee:

C:\clipper5>MENU ↵

UNIDAD IV CREACIÓN DE UNA BASE DE DATOS

4.1. Introducción al DBU de Clipper.

El DBU es una utileria de base de datos que permite crear, eliminar y modificar las estructuras de base de datos, además de crear base de datos, mostrar sus registros, crear archivos índices, entre otras operaciones sobre base de datos.

Un *campo,* es el conjunto de caracteres consecutivos que componen un dato.

Un *registro*, es el conjunto de datos relacionados entre sí.

Un *archivo*, es el conjunto de registros relacionados entre sí.

Las operaciones más comunes en los archivos son: agregar o incluir datos (*altas*), eliminar o borrar datos (*bajas*), corrección de datos (*modificaciones*), mostrar datos (*listado*), localización de datos (*consultas*), mostrar informes de datos (*reportes*).

Al clasificar los datos por un orden definido, debe crear índices, que le ayudan a realizar las operaciones anteriores con mayor rapidez.

Para crear un índice debe de contar con un campo llave, que identifica de forma única, una relación entre los demás archivos o registros. Este campo puede ser de tipo numérico, carácter o fecha, pero no más de 2 tipos a la vez, el tipo que elija depende de la forma en que desea realizar sus búsquedas.

Los objetivos al crear un Sistema Manejador de Base de datos, son los siguientes:

- *Redundancia de datos,* repetir con frecuencia los datos.
- *Dificultad al accesar a los datos,* cuando tiene los datos pero no tiene programas para usar esa información.
- *Problemas de seguridad,* cuando existen demasiadas usuarios del sistema y debe controlar el acceso a la información de personal no autorizado.
- *Inconsistencia,* cuando almacena los datos en los campos que no corresponde.

4.1.1. Uso del DBU para crear una base de datos.

Para crear una base de datos en Clipper, debe escribir:

Sintaxis: DBU <nombre de la base de datos>

El nombre de la base de datos es opcional, sino escribe algún nombre, se abre la pantalla principal del DBU y aparece:

Creación de Campos para la base de datos:

Para crear los campos presione la tecla F3(Crear) del menú elija Fichero, aparece el cuadro:

La columna *Nombre campo,* le permite establecer el identificador para el campo.

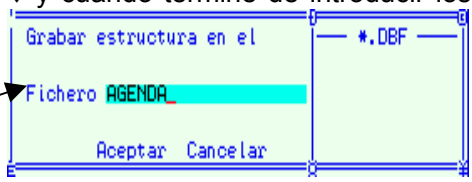
La columna Tipo, le permite establecer 1 de los 7 tipos de datos de clipper, por omisión aparece Carácter, si desea cambiar el tipo de dato presione la barra espaciadora.

La columna Anchura, le permite establecer el ancho del campo. Es decir, si el campo es de tipo carácter y el ancho es 25, este campo aceptara 25 caracteres.

La columna Decimal, permite que los tipos de datos numéricos puedan manejar punto decimal.

Cuando desee agregar otro campo presione ↓ y cuando termine de introducir los campos y desee salir presione la tecla F4(Grabar), aparece el cuadro:

En *Fichero*, escriba el nombre del archivo (DBF), presione ↵ y después en Aceptar ↵.

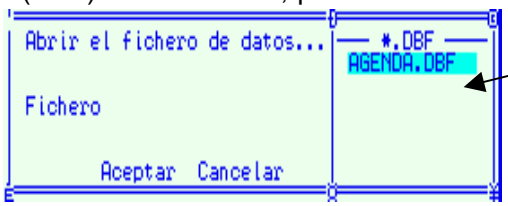


Salir del DBU.

Presione la tecla Esc, hasta que aparezca un mensaje que le preguntará ¿salir de la aplicación? (S/N), presione *S* para salir y *N* para permanecer en el DBU.

Abrir una base de datos existente:

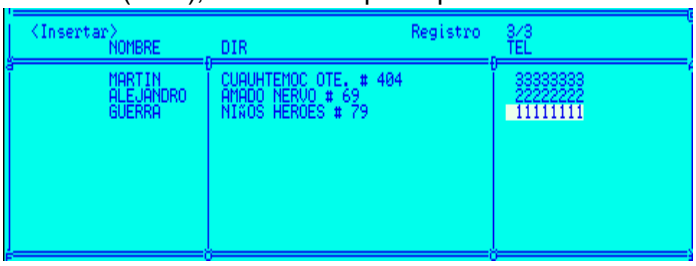
Verifique que el archivo (DBF) no este activo, presione la tecla F2 (Abrir) del menú elija Fichero, aparece el cuadro:



De la lista, elija el archivo (DBF) que desea abrir, después en Aceptar ↵.

Introduzca registros (datos) a su base de datos:

Si no esta activo el archivo (DBF), ábralo. Después presione la tecla F5 (Ver) del menú elija Fichero, aparece el cuadro:



Introduzca los datos correspondientes a cada campo, para introducir otro dato presione la tecla ↵. Sino alcanza a visualizar todos los campos de la base de datos, presione la tecla →. Si desea agregar otro registro presione la tecla ↓. Cuando desee salir de introducir datos (registros) presione la tecla Esc. No se preocupe de guardar los datos, Clipper automáticamente graba los datos.

Modifique, agregue o borre registros (datos) de la base de datos:

Debe verificar que el archivo (DBF) este activo, de lo contrario ábralo. Colóquese en el apartado Ficheros, elija el archivo (DBF) deseado, presione F5 (Ver) del menú elija Fichero, aparecerán los campos con los registros.

Para *modificar* los registros(datos), posicione el cursor en el dato deseado presione ↵, aparecerá el cursor, comience a escribir el nuevo dato. Si se sobreescribe active Insert para evitar esta acción.

Para *agregar* más registros, posicionece en el ultimo registro y presione la tecla ↓, agregue los datos deseados.

Para *borrar* un registro, posicionece en el registro a borrar y presione la tecla Supr. Cabe mencionar que este borrado, es solo lógico y en la pantalla superior ……, aparece un mensaje que le indica <Borrado>.

Cuando termine de modificar, agregar o borrar registros presione la tecla Esc.

Modifique, agregue o borre campos de la base de datos:

Debe verificar que el archivo (DBF) este activo, de lo contrario ábralo. Colóquese en el apartado Ficheros, elija el archivo (DBF) deseado, presione F3 (Crear) del menú elija Fichero, se mostrará el cuadro:

Elija el campo a *modificar,* presione ↵, aparecerá el cursor, realice la modificación que requiera.

Si desea *agregar* más campos a la estructura, colóquese al final de ella y presione la tecla ↓.

Si desea *borrar* un campo, colóquese en el campo a borrar y presione la tecla Supr.

Cuando termine de realizar los cambios deseados a su estructura, presione la tecla F4 (Grabar) y del menú elija estructura, le muestra:

De la lista, elija el archivo (DBF) al cuál corresponde la estructura y después en Aceptar ↵.

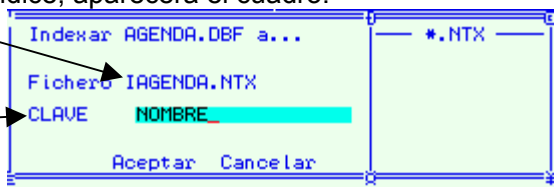
4.1.2. Creación de índices para una base de datos.

Verifique que el fichero (DBF) este activo, de lo contrario ábralo. Colóquese en al apartado Ficheros, elija el archivo deseado, presione la tecla F3 (Crear) del menú elija Indice, aparecerá el cuadro:

En *Fichero*, escriba el nombre del archivo índice (no mayor a 8 caracteres), después ↵.

En *Clave*, es el campo por el cuál se va a realizar la indexación (ordenación) de sus datos, después ↵.

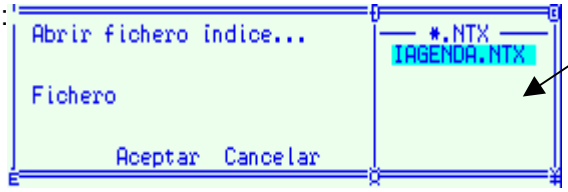
Por ultimo en *Aceptar* ↵.



Aparecerá en la pantalla un mensaje que le indica Fichero indexado.

Apertura y visualización del contenido del índice:

Para *abrir* el índice, primeramente debe verificar que el archivo índice (NTX) este activo, de lo contrario ábralo. Colóquese en el apartado Indices, elija el archivo índice deseado, presione la tecla F2 (Abrir) del menú elija Indice, aparecerá el cuadro:



De la lista, elija el archivo índice (NTX) deseado, después en Aceptar ↵.

Para *ver* el contenido del índice, primeramente debe verificar que el archivo índice (NTX) este activo, de lo contrario ábralo. Colóquese en el apartado Indices, elija el archivo índice deseado, presione la tecla F3 (Crear) del menú elija Indice, aparece un cuadro que le muestra el nombre del archivo (DBF), el nombre del archivo índice (NTX) y la clave por la cuál va a indexar, presione la tecla ↵. Ahora presione la tecla F5 (Ver) y del menú elija Fichero, aparece un cuadro con los datos indexados. Si desea salir, no olvide presionar la tecla Esc.

UNIDAD V ESTRUCTURAS DE CONTROL DE CLIPPER.

5.1. Sentencia IF.

IF <lCondición1>
<sentencias>...
[ELSEIF <lCondición2>]
<sentencias>...
[ELSE]
<sentencias>...
END[IF]

La estructura de control IF actúa dividiendo la ejecución a las sentencias situadas después de la primera condición que resulte verdadera (.T.), en la sentencia IF o en alguna de las ELSEIF. La ejecución continúa hasta que se encuentre la siguiente sentencia ELSEIF, ELSE o ENDIF.

Las estructuras IF...ENDIF pueden anidarse dentro de otras estructura IF...ENDIF y otras estructuras de control. Estas estructuras, no obstante, deben estar correctamente anidadas.

Argumentos: ***<lCondición>*** es una expresión lógica de control. Si resulta verdadera (.T.), todas las sentencias siguientes se ejecutan hasta que se encuentra una sentencia ELSEIF, ELSE o ENDIF.

✍ El ejemplo evalúa un número de condiciones utilizando una estructura IF ELSEIF...ENDIF :

```
LOCAL nNumero := 0
IF nNumero < 50
     ? "Menor que 50"
ELSEIF nNumero = 50
      ? "Igual a 50"
ELSE
      ? "Mayor que 50"
ENDIF
```

5.2. Instrucción DO CASE.

DO CASE
CASE <lCondición1>
<sentencias>...
[CASE <lCondición2>]
<sentencias>...
[OTHERWISE]
<sentencias>...
END[CASE]

DO CASE...ENDCASE es una estructura de control que ejecuta uno de varios posibles bloques de sentencias, dependiendo del resultado de evaluar las condiciones asociadas. Divide la ejecución a las sentencias situadas después de la primera condición, que resulte verdadera. La ejecución prosigue hasta que se encuentra el siguiente CASE, OTHERWISE o ENDCASE.

Si ninguna de las condiciones CASE resulta verdadera, las sentencias que siguen a la sentencia OTHERWISE se ejecutan hasta la sentencia ENDCASE correspondiente. Si se omite una sentencia OTHERWISE, el control pasa a la primera sentencia que sigue a la sentencia ENDCASE correspondiente.

Es posible anidar cualquier número de sentencias dentro de una sentencia DO CASE, incluyendo otras estructuras de control (es decir, DO WHILE y FOR).

Argumentos: ***CASE <lCondición>*** define el bloque de sentencias que se van a ejecutar si <lCondición> resulta verdadera (.T.). ***OTHERWISE*** define un bloque de sentencias que se van a ejecutar si ninguna de las condiciones CASE especificadas resulta verdadera (.T.).

✍ El ejemplo utiliza DO CASE en una estructura de menús para bifurcar el control según la selección del usuario:

```
@ 10,10 SAY “Elija su opcion:” GET nOpcion
   READ
DO CASE
CASE nOpcion = 0
  RETURN
CASE nOpcion = 1
```

```
  OpcionUno()
CASE nOpcion = 2
  OpcionDos()
ENDCASE
```

5.3. Ciclo DO WHILE, WHILE Y FOR.

[DO] WHILE <lCondición>
<sentencias>...
[EXIT]
<sentencias>...
[LOOP]
<sentencias>...
END[DO]

DO WHILE...ENDDO es una estructura de control de secuencia que ejecuta repetitivamente un bloque de sentencias mientras <lCondición> sea verdadera. Mientras esta condición resulte verdadera, el control pasa a la estructura y continúa hasta encontrar una sentencia EXIT, LOOP o ENDDO. ENDDO devuelve el control a la sentencia DO WHILE y el proceso se repite. Cuando la condición es falsa, finaliza la estructura DO WHILE y el control pasa a la sentencia inmediatamente siguiente a ENDDO.

EXIT permite terminar una estructura DO WHILE con una condición distinta a la condición DO WHILE original. LOOP, permite saltar a evaluar nuevamente la condición del de DO WHILE basadas en una condición intermedia y devuelve el control a la sentencia DO WHILE más reciente.

La estructura DO WHILE puede anidarse dentro de otras estructuras de control.

Argumentos: ***<lCondición>*** es la expresión de control lógica del bucle DO WHILE.

✍ El ejemplo muestra cómo puede utilizarse LOOP para establecer un tratamiento condicional:

```
DO WHILE <lCondición>
  <tratamiento inicial>...
  IF <Condición intermedia>
    LOOP
  ENDIF
  <tratamiento restante>...
ENDDO
```

✍ El ejem. muestra la utilización de DO WHILE para emular una repetición en una estructura de bucle:

```
LOCAL lMas := .T.
DO WHILE lMas
  <sentencias>...
  lMas := (<lCondición>)
ENDDO
```

✍ El ejem. utiliza un bucle DO WHILE para desplazarse secuencialmente por un fichero de b. de datos:

```
DO WHILE .NOT. EOF()
<sentencias>...
  SKIP
ENDDO
```

FOR <idContador> := <nInicio> TO <nFin>
[STEP <nIncremento>]
<sentencias>...
[EXIT]
<sentencias>...
[LOOP]
NEXT

FOR...NEXT es una estructura de control que ejecuta un bloque de sentencias un número de veces especificado. La estructura de control efectúa un bucle desde el valor inicial de <idContador> hasta el límite especificado por <nFin>, desplazándose a través del rango de valores de la variable de control con un incremento especificado mediante <nIncremento>.

Las construcciones FOR...NEXT pueden anidarse dentro de otras estructuras de control.

Argumentos: ***<idContador>*** es el nombre de la variable de control o contador del bucle. Si el <idContador> especificado no es visible o no existe, se crea una variable privada. ***<nInicio>*** es el valor inicial asignado a <idContador>. ***STEP <nIncremento>*** define la cantidad en que varía <idContador> para cada iteración del bucle. <nIncremento> puede ser positivo o negativo. Si no se especifica la cláusula STEP, <idContador> se incrementa en uno en cada iteración del bucle. ***EXIT*** bifurca el control incondicionalmente a la sentencia inmediatamente siguiente a la sentencia NEXT más próxima. ***LOOP*** separa el control del FOR ejecutado más recientemente.

✍ El ejemplo rellena la pantalla con un carácter, utilizando FOR anidados:

```
FOR i := 0 TO 79
 FOR j:=0 TO 24
  @ i, j SAY "*"
 NEXT
NEXT
```

✍ El ejemplo recorre una matriz en orden ascendente:

```
nLongMatriz := LEN(aMatriz)
FOR i := 1 TO nLongMatriz
  <sentencias>...
NEXT
```

✍Para recorrer una matriz en orden descendente:

```
nLongMatriz := LEN(aMatriz)
FOR i := nLongMatriz TO 1 STEP -1
  <sentencias>...
NEXT
```

5.4. Procedimientos frente a Funciones.

RETURN [<exp>]

RETURN finaliza un procedimiento, función o programa, devolviendo el control al procedimiento o función que la invocó. Al devolverse el control al procedimiento llamante, se liberan todas las variables privadas y locales declaradas en el procedimiento o función actual. En un procedimiento o función puede haber más de una sentencia RETURN. La función debe contener al menos una sentencia RETURN con un argumento.

Argumentos: ***<exp>*** es una expresión de cualquier tipo que da como resultado el valor de retorno para funciones de usuario. Si una función finaliza sin una sentencia RETURN, el valor de retorno es NIL.

✍ El ejem. muestra el formato general de la sentencia RETURN en un procedimiento y en una función:

```
PROCEDURE <idProcedimiento>
 <sentencias>...
RETURN

FUNCTION <idFuncion>
  <sentencias>...
RETURN <expDevuelto>
```

FUNCTION <idFunción> [(<idLista Parámetros>)]
[LOCAL <identificador> [[:= <inicializador>], ...]]
[STATIC <identificador> [[:= <inicializador>], ...]]
.
. <sentencias ejecutables>
.
RETURN <exp>

La sentencia FUNCTION declara una función definida por el usuario y una lista opcional de variables locales para recibir parámetros, que con frecuencia se denominan parámetros formales. Una función definida por el usuario es un subprograma compuesto por un conjunto de declaraciones y sentencias, que se ejecutan siempre que se hace referencia a <idFunción> seguido por un par de paréntesis de apertura y cierre. Una definición de función comienza con una sentencia FUNCTION y finaliza con RETURN.

Argumentos: ***<idFunción>*** es el nombre de la función definida por el usuario que debe declararse. Los nombres de función definidos por el usuario pueden tener cualquier longitud, pero sólo son significativos los

primeros 10 caracteres. Los nombres pueden contener cualquier combinación de caracteres, números o signos de subrayado, pero deben comenzar con un carácter. ***<idLista Parámetros>*** es la declaración de una o más variables de parámetro. Las variables especificadas en esta lista se han declarado como locales. ***LOCAL*** o ***STATIC*** declara y, opcionalmente, inicializa una lista de variables o matrices cuya visibilidad y tiempo de vida es la función actual.

Notas: Llamada a una función definida por el usuario: Utilice la misma notación para llamar a una función definida por el usuario que cuando efectúa una llamada a una función de CA-Clipper estándar:

<idFunción>([<lista argumentos>])

Parámetros: Un parámetro es un espacio reservado para un valor o una referencia.

Las funciones reciben parámetros en el orden que se les han pasado. En Clipper, el número de parámetros no tiene por qué coincidir con el número de argumentos pasados. Puede saltarse argumentos u omitirlos de la lista de argumentos. Un parámetro que no recibe un valor o la referencia se inicializa con NIL.

Los parámetros especificados en una función definida por el usuario pueden recibir argumentos pasados por valor o referencia. El método por defecto para expresiones y variables es por valor. Esto incluye variables que contienen referencias a matrices y objetos. Todas las variables excepto las de campo, cuando están precedidas por el operador pasar por referencia (@), se pasan por referencia. Las variables de campo no pueden pasarse por referencia y se pasan siempre por valor.

✍ Se muestra una función definida por el usuario que toma 2 números y regresa la suma de los 2 valores:

```
Result:=Suma(10,10)
FUNCTION Suma(num1,num2)
  LOCAL Result
  Result:=num1+num2
RETURN Result
```

PROCEDURE <idProcedimiento> [(<idLista parám>)]
[LOCAL <identificador> [[:= <inicializador>], ...]]
[STATIC <identificador> [[:= <inicializador>], ...]]
.
. <sentencias ejecutables>
.
[RETURN]

La sentencia PROCEDURE declara un procedimiento y una lista opcional de variables locales para recibir los parámetros pasados por la rutina invocante. Un procedimiento es un subprograma compuesto por un grupo de declaraciones y sentencias que se ejecutan cuando se escribe <idProcedimiento> seguido de un paréntesis inicial y final o mediante una sentencia DO. Una definición de procedimiento comienza con la sentencia PROCEDURE y finaliza con la sentencia RETURN.

Argumentos: ***<idProcedimiento>*** es el nombre del procedimiento que se declara. Los nombres de procedimiento pueden tener cualquier longitud, pero sólo son significativos los 10 primeros caracteres. Pueden contener cualquier combinación de letras, números o caracteres de subrayado, pero los subrayados iniciales están reservados. ***<idLista parám>*** es la declaración de una o más variables de parámetro. Las variables especificadas en esta lista se declaran como locales. ***LOCAL*** o ***STATIC*** declara y, opcionalmente, inicializa una lista de variables o matrices de visibilidad y duración limitadas al procedimiento actual. ***<identificador>, <lista identificadores>*** es una etiqueta o etiquetas utilizadas como el nombre de la variable o matriz que se va a crear. ***RETURN*** devuelve el control al procedimiento o función invocante. Si no se especifica RETURN, el control se devuelve a la rutina invocante al terminar la definición del procedimiento.

Notas: Llamada a un procedimiento: En CA-Clipper, un procedimiento puede llamarse de dos formas. La primera, y preferible, es la convención de llamada de funciones. Este método para invocar un procedimiento es idéntico al que se utiliza para invocar una función, en una línea individual:

<idProcedimiento>([<lista argumentos>])

El segundo método es la convención de llamada mediante el mandato DO...WITH. Los dos métodos difieren sólo en la forma de paso de parámetros por defecto. La convención funcional de llamada pasa las variables por valor, mientras que la convención de llamada por mandato los pasa por referencia.

Parámetros: Un parámetro es un lugar para un valor o una referencia. En CA-Clipper, existen dos formas de recibir parámetros: puede declararse una lista de nombres de variables locales como parte de la declaración PROCEDURE (parámetros formales).

Los procedimientos reciben los parámetros en el mismo orden en el que se pasan. En CA-Clipper, el número de parámetros no tiene que ser igual al número de argumentos pasados. Los argumentos pueden omitirse o dejarse al final de la lista de argumentos.

Los parámetros especificados en un procedimiento pueden recibir argumentos pasados por valor o por referencia. El método por defecto para las expresiones y variables depende de la convención de llamada. En la convención de llamada de funciones, el método por defecto para pasar expresiones y variables es por valor, incluyendo variables que contienen referencias a matrices y objetos. En la convención de llamada de mandatos, el método por defecto para pasar variables es por referencia.

✍ El ejemplo muestra un procedimiento empleado para borrar una parte de la pantalla:

```
LIMPIA(10,0,18,79)
 PROCEDURE LIMPIA(esqsup1, esqsup2, esqinf1,esqinf2 )
  @ esqsup1, esqsup2 CLEAR TO esqinf1,esqinf2
 RETURN
```

UNIDAD VI HERRAMIENTAS Y COMANDOS BASICOS DE CLIPPER

COMANDOS PARA ENTRADA Y SALIDA DE DATOS EN PANTALLA

CLEAR o CLS

Es un mandato de borra la pantalla, libera los objetos GET pendientes y coloca el cursor en la fila y columna cero. Si se especifica la cláusula SCREEN, los objetos Get no se liberan.

@ <nArriba>, <nIzquierda> CLEAR TO <nAbajo>, <nDerecha>

@...CLEAR borra una zona rectangular de la pantalla, llenando esa región con caracteres de espacio y utilizando los valores de color estándar actuales. Cuando @...CLEAR borra la región indicada, el cursor se sitúa en la esquina superior de la región en <nArriba>+1 y <nIzquierda>+1.

Argumentos: ***<nArriba> y <nIzquierda>*** definen las coordenadas de la esquina superior izquierda. TO ***<nAbajo>, <nDerecha>*** definen las coordenadas de la esquina inferior derecha de la zona de la pantalla donde se ejecuta CLEAR. Si no se especifica la cláusula TO, se toman las coordenadas por defecto de MAXROW() y MAXCOL().

✍ El ejemplo borra la pantalla desde 10,10 a 20,40:

```
@ 10, 10 CLEAR TO 20, 40
```

@ <nFila>, <nColumna> SAY <exp> [PICTURE <cPatróndeFormatoSay>] [COLOR <cCadenaColor>]

Es un mandato de pantalla completa que dirige el resultado de <exp> a la pantalla o a la impresora, en las coordenadas especificadas de fila y columna. Puede formatear este resultado con una cláusula PICTURE.

Argumentos: ***<nFila> y <nColumna>*** son las coordenadas de fila y columna de la visualización. ***SAY <exp>*** muestra el resultado de una expresión de caracteres fechas, valores lógicos o números. ***PICTURE <cPatróndeFormatoSay>*** controla el formato de la visualización de <exp>. CA-Clipper dispone de dos mecanismos, funciones y plantillas para el control de formato. Las funciones se aplican al mandato SAY completo mientras que las plantillas formatean los caracteres posición a posición. ***COLOR <cCadenaColor>*** define el color de visualización de <exp>. Si especifica una selección de colores literal, debe escribirla entre comillas. Los mandatos combinados @...SAY...GET requieren dos cláusulas COLOR: una para especificar los colores de SAY y otra para los colores de GET.

Símbolos de Plantilla para SAY y PICTURE()

Plantilla	Acción	Plantilla	Acción
A, N, X, 9, #	Muestra dígitos de cualquier tipo de datos	$	Muestra un signo de dólar en lugar de un espacio inicial
L	Muestra los valores lógicos como "T" o "F"	*	Muestra un * en lugar de un espacio en blanco inicial
Y	Muestra los valores lógicos como "Y" o "N"	.	Especifica una posición de punto decimal
!	Convierte en mayúsculas los caracteres alfabéticos	,	Especifica una posición de coma decimal

✍ El ejemplo utiliza un mandato @...SAY con una cláusula PICTURE para mostrar un resultado formateado.

```
nIngresoNeto = 7125.50
nPerdidaNeta = -125.50
```

```
cTelefono = "2134567890"
cNombre = "Julia Gómez"
@ 1, 1 SAY nIngresoNeto PICTURE "@E 9.999,99"          // Resultado: 7.125,50
@ 2, 1 SAY nPerdidaNeta PICTURE "@)"                   // Resultado: (125,50)
@ 3, 1 SAY cTelefono PICTURE "@R (999)999-9999"        // Resultado: (213)456-7890
@ 4, 1 SAY cNombre PICTURE "@!"                        // Resultado: JULIA GOMEZ
```

@ <nFila>, <nColumna> SAY<exp> [PICTURE<cPatróndeFormatoSay>] COLOR<cCadenaColor>]]
GET <idVar> [PICTURE <cPatróndeFormatoGet>] [COLOR <cCadenaColor>]
[RANGE* <fnMínimo>, <fnMáximo>] | [VALID <lPostCondición>]

El mandato @...GET crea un nuevo objeto GET, muestra su valor en pantalla y lo añade a la variable.

Cada objeto GET tiene una variable asociada, <idVar>. Esta variable puede tener cualquier clase de almacenamiento, incluyendo campos de base de datos, variables privadas, públicas, locales o estáticas.

Puede validar utilizando las cláusulas VALID o RANGE*. VALID especifica una condición que debe cumplirse antes de que el cursor pueda salir del objeto GET. Si el resultado es verdadero, el cursor puede salir; en caso contrario, el cursor permanece en el objeto GET. RANGE* especifica un rango de valores aceptables para números. Si el valor introducido por el usuario no se encuentra en el rango especificado, el cursor no puede salir.

Símbolos de la Plantilla GET PICTURE

Plantilla	Acción	Plantilla	Acción
A	Sólo permite caracteres alfabéticos	Y	Sólo permite Y o N
N	Sólo permite caracteres alfabéticos y numéricos	!	Convierte en mayúsculas los caracteres alfabéticos
X	Permite cualquier carácter	$	Muestra un signo de dólar en las posiciones de los espacios iniciales de un valor numérico
9	Permite dígitos de cualquier tipo de datos incluyendo signos de caracteres numéricos	*	Muestra un asterisco en lugar de un espacio en blanco en un valor numérico
#	Permite dígitos, signos y espacios de cualquier tipo de datos	.	Muestra un punto decimal
L	Sólo permite T, F, Y o N	,	Muestra una coma

Argumentos: ***<nFila> y <nColumna>*** especifican las coordenadas de fila y columna de la operación. Si existe una cláusula SAY, <nFila> y <nColumna> especifican las coordenadas de la cláusula SAY y GET aparece a la derecha de la salida de SAY. ***SAY <exp>*** muestra el valor de <exp> en las coordenadas especificadas. Si especifica la cláusula ***PICTURE <cPatróndeFormatoSay>***,<exp> se formatea según las reglas de los patrones de salida SAY. ***GET <idVar>*** especifica el nombre de la variable asociada al mandato GET. <idVar> puede ser de cualquier clase de almacenamiento. Si <idVar> contiene un valor de matriz, debe indicar uno o mas subíndices. El valor actual de <idVar> se muestra en las coordenadas de GET. Este valor debe ser caracteres, fechas, números o valores lógicos. No se permiten matrices, NIL, bloques de código ni cadenas nulas. ***PICTURE <cPatróndeFormatoGet>*** especifica el formato de pantalla y las reglas de edición del mandato GET. ***COLOR <cCadenaColor>*** define los colores seleccionados para el objeto GET actual. Si desea especificar valores literales de colores, debe escribirlos entre comillas. Si especifica un solo color, determina el color seleccionado y no seleccionado del objeto GET. En un mandato combinado @...SAY...GET , se requieren dos cláusulas COLOR para especificar los colores de SAY y de GET: uno para SAY y otro para GET. **VALID <lPostCondición>** especifica una expresión que debe satisfacerse antes de que el cursor pueda salir del objeto GET durante una operación READ. ***RANGE* <fnMínimo>, <fnMáximo>*** especifica un rango de valores permitidos como entrada del mandato GET.

✍ El ejemplo muestra la utilización de una cláusula VALID para validar los datos de un GET:

```
LOCAL nNumero := 0
@ 10, 10 SAY "Introduzca un número:" ;
  GET nNumero VALID nNumero > 0
```

El ejemplo utiliza la función @K para sugerir un valor de entrada por defecto, pero lo borra si la primera tecla pulsada no es una tecla del cursor o Intro:

```
LOCAL cFichero := "Cuentas"
@ 1, 1 SAY "Introduzca el fichero" GET cFichero PICTURE "@K"
READ
```

READ

Es un mandato que activa una edición de pantalla completa usando todos los objetos Get creados y añadidos a la GetList actual. Si hay algún procedimiento de formato activo, READ ejecuta ese procedimiento antes de entrar en el modo de edición de pantalla completa.

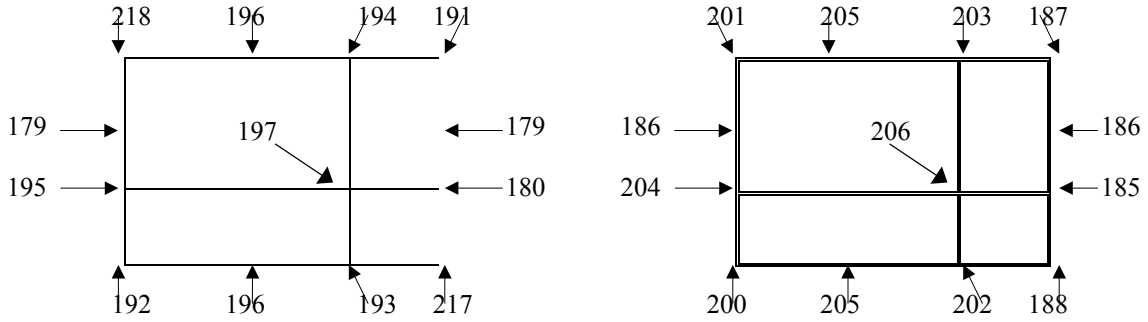
✍ El ejemplo define varios objetos GET y, a continuación, los lee:

```
cVar1 := cVar2 := cVar3 := SPACE(10)
@ 10, 10 SAY "Variable uno:" GET cVar1
@ 11, 10 SAY "Variable dos:" GET cVar2
@ 12, 10 SAY "Variable tres:" GET cVar3
READ
```

@ <nArriba>, <nIzquierda>, <nAbajo>, <nDerecha> BOX <cCadenaCuadro> COLOR <cCadenaColor>

@...BOX dibuja un cuadro en la pantalla utilizando caracteres configurables de borde y relleno. @...BOX dibuja el cuadro utilizando <cCadenaCuadro> empezando en la esquina superior izquierda y dibujando el cuadro en el sentido de las agujas del reloj, rellenando toda la zona con el noveno carácter. Si no se especifica este carácter, no se dibuja la zona interior del cuadro. El texto y los colores existentes permanecen invariables.

Argumentos: ***<nArriba>, <nIzquierda>, <nAbajo>, <nDerecha>*** definen las coordenadas del cuadro. @...BOX dibuja un cuadro utilizando valores de filas comprendidos entre 0 y MAXROW() y valores de columnas entre 0 y MAXCOL(). ***BOX <cCadenaCuadro>*** define una cadena de 8 caracteres de borde y un carácter de relleno. Si <cCadenaCuadro> se especifica como un único carácter, ese carácter dibuja todo el cuadro. ***COLOR <cCadenaColor>*** define el color en pantalla del cuadro dibujado.. Si desea especificar una selección de color literal, debe escribirla entre comillas.



✍ El ejemplo muestra 3 cajas; 1 sin relleno y las otras 2 con relleno.

```
SET COLOR TO R
@ 1, 0, 23, 79 BOX "218 196 191 179 217 196 192 179 “          //Oprima Alt con esos números.
SET COLOR TO B+/W+
@ 5, 10, 19, 70 BOX "201 205 187 186 188 205 200 186 219”    //Oprima Alt con esos números.
SET COLOR TO G+/W+
@ 10, 20, 14, 60 BOX "218 196 191 179 217 196 192 179 178” //Oprima Alt con esos números.
```

INKEY(<nSegundos>)

Es una función de teclado que lee la tecla siguiente contenida en la memoria intermedia del teclado y devuelve un valor que representa esa tecla. Este valor se guarda también internamente y puede recuperarse por medio de LASTKEY(). INKEY() devuelve un valor numérico comprendido entre -39 y 386

Argumentos: ***<nSegundos>*** Especifica el número de segundos que INKEY() espera a que se pulse una tecla. Si se especifica cero, el programa se detiene hasta que se pulse una tecla. Si se omite <nSegundos> INKEY() no espera a que se pulse una tecla.

✍ El ejemplo recoge una tecla del teclado y muestra el valor de su carácter seguido por el valor de INKEY():

```
#include "Inkey.ch"
LOCAL nCodigoInkey := 0
DO WHILE LASTKEY() != K_ESC
  ? "Pulse una tecla: "
nCodigoInkey := INKEY(0)
```

```
   ?? "Carácter:", CHR(nCodigoInkey),;
      "Código INKEY():", LTRIM(STR(nCodigoInkey))
ENDDO
RETURN
```

LASTKEY()

Es una función de teclado que informa del valor INKEY() de la última tecla recogida de la memoria intermedia del teclado por la función INKEY(). LASTKEY() conserva su valor actual hasta que se recoge otra tecla de la memoria intermedia del teclado. Lastkey() permite:

- Determinar la tecla que termina un READ.
- Determinar la tecla que provoca la salida del objeto GET actual en una función definida por el usuario, invocada por una cláusula VALID.
- Identificar una tecla de excepción en la función de usuario de ACHOICE(), DBEDIT() o MEMOEDIT().

✍ El ejemplo espera un segundo para que oprima cualquier tecla diferente a <Esc> o <Intro>.

```
DO WHILE ((LASTKEY()!=27) .AND. (LASTKEY()!=13))
   INKEY(1)
ENDDO
```

SET COLOR TO [[<estándar>] [,<resaltado>] [,<borde>] [,<fondo>][,<sinseleccionar>]] | (<cCadenaColor>)

Este mandato especifica una lista de valores de colores para los cinco tipos de actividad de dibujo de pantalla. Cada ajuste o valor es un par de colores de primer plano y fondo separados por un carácter de barra (/). El color del primer plano define el color de los caracteres que se muestran en pantalla. El fondo lo define el color que aparece detrás del carácter barra ("/"). Los espacios y caracteres que no se muestran aparecen sólo como fondo. Cada color puede especificarse usando una letra o un número, pero no ambos a la vez en un mismo parámetro. La utilización de los números no se recomienda.

Argumentos: ***<estándar>*** es el color en que se dibujan todos los mandatos y funciones de consola, pantalla completa e interfaz cuando se muestran en la pantalla. Esto incluye mandatos como @...PROMPT, @...SAY y ?, y funciones como ACHOICE(). ***<resaltado>*** es el color en que se muestran las visualizaciones resaltadas. ***<borde>*** es el color en que se muestra el área que rodea a la pantalla y en la cual no puede escribirse. ***<fondo>*** no está admitido actualmente por ninguna de las máquinas con controladores de Computer Associates. Se ofrece sólo por motivos de compatibilidad. ***<sinseleccionar>*** es un par de colores en que se muestran los GET sin enfoque de entrada y las opciones de menú no disponibles. ***<cCadenaColor>*** es una cadena de caracteres, entre paréntesis, que contiene la configuración de color.

TABLA DE COLORES

COLOR	LETRA	NUMERO	COLOR	LETRA	NUMERO	COLOR	LETRA	NUMERO
Negro	N	0	Marrón	GR	6	Rojo intenso	R+	12
Azul	B	1	Blanco	W	7	Magenta intenso	RB+	13
Verde	G	2	Gris	N+	8	Amarillo	GR+	14
Cían	BG	3	Azul intenso	B+	9	Blanco intenso	W+	15
Rojo	R	4	Verde intenso	G+	10	Vídeo inverso	I	
Magenta	RB	5	Cían intenso	BG+	11			

COMANDOS PARA LA CREACIÓN DE MENÚS

SET MESSAGE TO <nRenglón> <CENTER>

Es un mandato de menú que define la fila de la pantalla en la que se muestran los mensajes @...PROMPT. Cuando se llama a un programa, el valor por defecto de fila de mensaje es cero: supresión de todos los mensajes definidos. Los mensajes aparecen en <nFila>, columna 0 a menos que se utilice la opción CENTER.

SET WRAP ON/OFF

Este mandato activa o desactiva el desplazamiento cíclico de la barra resaltada en un menú @...PROMPT desde el primer elemento del menú al último y viceversa. Cuando se establece SET WRAP ON y la barra resaltada se encuentra en la última opción del menú, Flecha derecha o Flecha abajo desplazan la zona resaltada al primer elemento del menú. De igual modo, cuando la barra de selección está sobre la primera opción del menú, Flecha izquierda o Flecha arriba desplazan la zona resaltada al último elemento del menú.

Cuando se establece SET WRAP OFF, la pulsación de Flecha arriba o Flecha izquierda desde el primer elemento de menú o de Flecha abajo o Flecha derecha desde el último no tiene ninguna consecuencia.

@ <nFila>, <nColumna> PROMPT <cElemMenú> [MESSAGE <cExpresión>]

@...PROMPT es la parte de visualización del sistema de menús de barra de CA-Clipper. Cada mandato @...PROMPT dibuja un elemento de menú en el color estándar actual y define un MESSAGE asociado que se muestra en la línea especificada por el valor establecido en SET MESSAGE. El menú se invoca a continuación con el mandato MENU TO. Puede especificar elementos de menú en cualquier orden y con cualquier configuración de posiciones de filas y columnas. MENU TO, no obstante, se desplaza por la lista actual de elementos de menú en el orden en que están definidos. Puede definir como máximo 4.096 elem. en cada menú.

Argumentos: ***<nFila> y <nColumna>*** son las coordenadas de fila y columna del elemento de menú mostrado. Los valores de fila pueden estar comprendidos entre cero y MAXROW(), y los valores de columna entre cero y MAXCOL(). ***PROMPT <cElemMenú>*** es la cadena del elemento del menú que se va a mostrar. ***MESSAGE <cExpresión>*** define el mensaje que se muestra cuando se selecciona el elemento de menú actual. Puede utilizarse opcionalmente un bloque de código que evalúe una expresión de caracteres.

MENU TO <idVar>

Este mandato activa un menú iluminado para las opciones definidas con la orden @ ...PROMPT y asigna la selección del usuario a la variable especificada como un valor numérico. Permite emplear las teclas ← ↑ ↓ → Inicio, Fin, AvPag, RePag para colocar el cursor en una opción del menú. También permite el empleo de la primera letra de la opción para su selección. <Esc> interrumpe el menú. <Intro> selecciona la opción en donde se encuentra el cursor.

✍ El ejemplo crea un menú de barra vertical sencillo, con los mensajes centrados en la línea 23.

```
LOCAL nSeleccion := 2
SET WRAP ON
SET MESSAGE TO 23 CENTER
@ 6, 10 PROMPT "Añadir"  MESSAGE "Nueva Cuenta"
@ 7, 10 PROMPT "Editar "  MESSAGE "Cambiar Cuenta"
@ 9, 10 PROMPT " Salir  "  MESSAGE "Volver al DOS"
MENU TO nSeleccion
DO CASE
CASE nSeleccion = 1
  NuevaCuenta()
CASE nSeleccion = 2
  CambioCuenta()
CASE nSeleccion = 3
  QUIT
ENDCASE
RETURN
```

COMANDOS PARA DATOS DE TIPO CARACTER

ALLTRIM(cCadena)

Esta función suprime los espacios en blanco iniciales y finales de una cadena de caracteres.

Argumentos: ***<cCadena>*** es la expresión de caracteres que se va a recortar.

✍ El ejem. crea una cadena con espacios iniciales y finales en blanco y luego los elimina con ALLTRIM():

```
cCadena := SPACE(10) + "cadena" + SPACE(10)
? LEN(cCadena)                  //Resultado: 26
? LEN(ALLTRIM(cCadena))         //Resultado: 6
```

[R]TRIM(<cCadena>)

Es una función que formatea cadenas de caracteres. Sirve para borrar espacios finales al concatenar cadenas, lo que ocurre con mucha frecuencia en los campos de base de datos que se almacenan con un formato de anchura fija. Por ejemplo, RTRIM() puede utilizarse para concatenar los campos que contienen el nombre y el apellido para formar una cadena con el nombre completo.

Argumentos: ***<cCadena>*** es la cadena de caracteres que se va a copiar sin espacios finales.

✍ El ejem., RTRIM() formatea los campos Ciudad, Provincia y Código postal para etiquetas y circulares:

```
USE Clientes INDEX NombreCli NEW
SEEK "Rosa"
? RTRIM(cCiudad) + ", " + RTRIM(cProvincia) + " " + cCodigoPostal // Resultado: Oviedo, Asturias 43066
```

LTRIM(<cCadena>)

Es una función que elimina los espacios en blanco iniciales de una cadena de caracteres. Permite formatear cadenas de caracteres con espacios iniciales en blanco. Estas pueden ser, por ejemplo, números que se convierten en una cadena de caracteres utilizando STR().

Argumentos: ***<cCadena>*** es la cadena de caracteres de la que desea eliminar los espacios iniciales en blanco.

✍ El ejemplo muestra el resultado del uso de la función LTRIM():

```
LOCAL Nombre:=”  Mar Rojo”
? LTRIM(Nombre)              // Resulta sin espacio inicial, Mar Rojo
```

LOWER(<cCadena>)

Es una función de caracteres que convierte cadenas en mayúsculas y mixtas en cadenas en minúsculas. Está relacionada con UPPER() que convierte cadenas en minúsculas y mixtas en cadenas en mayúsculas.

Se utiliza normalmente para formatear una cadena de caracteres para su visualización en pantalla.

Argumentos: ***<cCadena>*** es la cadena de caracteres que desea convertir en minúsculas.

✍ Estos ejemplos muestran los resultados de diferentes usos de LOWER():

```
? LOWER("CADENA")                    // Resultado: cadena
? LOWER("1234 CARACTERES = ")    // Resultado: 1234 caracteres =
```

UPPER(<cCadena>)

Es una función de caracteres que convierte cadenas en minúsculas y mixtas en cadenas en mayúsculas. Está relacionada con LOWER() que convierte cadenas en mayúsculas y mixtas en cadenas en minúsculas. UPPER() se utiliza normalmente para formatear cadenas de caracteres con fines de visualización.

Argumentos: ***<cCadena>*** es la cadena de caracteres que se va a convertir.

✍ El ejemplo muestra el resultado de UPPER():

```
? UPPER("una cadena")          // Resultado: UNA CADENA
? UPPER("123 carac = <>")      // Resultado: 123 CARAC = <>
```

LEN(<cCadena>)

Es una función de caracteres que devuelve la longitud de una cadena de caracteres. Si la cadena de caracteres es una cadena nula (""), LEN() devuelve cero.

Argumentos: ***<cCadena>*** es la cadena de caracteres que se va a contar.

✍ Estos ejemplos muestran la utilización de LEN() con varios argumentos:

```
? LEN("cadena de caracteres")  // Resultado: 20
? LEN("")                      // Resultado: 0
? LEN(CHR(0))                  // Resultado: 1
LOCAL aPrueba[10]
? LEN(aPrueba)                 // Resultado: 10
```

SUBSTR(<cCadena>, <nInicio>, [<nCaracteres>])

Es una función de caracteres que extrae una subcadena de otra cadena de caracteres o campo memo.

Argumentos: ***<cCadena>*** es la cadena de caracteres de la que va a extraerse una subcadena. Puede tener hasta 65.535 (64K) bytes, el tamaño máximo de cadena en CA-Clipper. ***<nInicio>*** es la posición inicial en <cCadena>. ***<nCaracteres>*** es el número de caracteres que hay que extraer. Si se omite, la subcadena empieza en <nInicio> y continúa hasta el final de la cadena. Si <nCaracteres> es mayor que el número de caracteres desde <nInicio> hasta el final de <cCadena>, los caracteres adicionales se ignoran.

✍ Estos ejemplos extraen el nombre y apellido de una variable:

```
cNombre:= "Juan Pedralbes"
```

```
? SUBSTR(cNombre, 1, 4)                  // Resultado: Juan
? SUBSTR(cNombre, 6)                     // Resultado: Pedralbes
? SUBSTR(cNombre, LEN(cNombre) + 2)      // Resultado:  cadena nula
? SUBSTR(cNombre, -9)                    // Resultado: Pedralbes
```

CHR(<nCódigo>)

Esta función devuelve el carácter correspondiente al código ASCII especificado por <nCódigo>.
Argumentos: ***<nCódigo>*** es un código ASCII comprendido entre 0 y 255.

✍ Estos ejemplos muestran la utilización de CHR() con distintos argumentos:

```
? CHR(72)                // Resultado: H
? CHR(ASC("A") + 32)     // Resultado: a
? CHR(7)                 // Resultado: suena la señal acústica
```

COMANDOS PARA FECHA, TIEMPO, SONIDO, DOCUMENTACIÓN Y DEL DOS.

SET CENTURY [ON|OFF]

Modifica el formato de fecha para incluir u omitir los dígitos de siglo. SET CENTURY ON cambia el formato de fecha para que contenga cuatro dígitos para el año. Con el formato de fecha configurado para admitir cuatro dígitos para el año, los valores de fecha se muestran con los cuatro dígitos del año y pueden introducirse fechas de cualquier siglo. SET CENTURY OFF cambia el formato de fecha para que contenga sólo dos dígitos para el año.

Argumentos: ***ON*** permite la escritura y visualización de los dígitos de siglo en las fechas. ***OFF*** suprime la escritura y visualización de los dígitos de siglo en las fechas.

✍ Este ejemplo muestra los resultados de un mandato SET CENTURY sencillo:

```
SET CENTURY OFF
? DATE()             // Resultado: 15/09/90
SET CENTURY ON
? DATE()             // Resultado: 15/09/1990
```

DATE()

Es una función de conversión que permite inicializar variables de memoria con la fecha actual, comparar otros valores de fecha con la fecha actual y ejecutar cálculos de fecha correspondientes a la fecha actual. El formato de visualización de fechas se controla con el mandato SET DATE. El formato por defecto es mm/dd/aa.

✍ Estos ejemplos muestran diversos resultados de la utilización de la función DATE():

```
? DATE()                 // Resultado: 09/01/90
? DATE() + 30            // Resultado: 10/01/90
? DATE() - 30            // Resultado: 08/02/90
fFecha := DATE()
? CMONTH(fFecha)         // Resultado: Septiembre
```

CDOW(<dExp>)

Esta función convierte un valor de fecha en un día de la semana en formato de cadena de caracteres

✍ Los ejemplos siguientes ilustran CDOW():

```
? DATE()                 // Resultado: 09/01/90   ? CDOW(DATE())                // Resultado: Viernes
? CDOW(DATE() + 7)       // Resultado: Viernes    ? CDOW(CTOD("06/12/90"))     // Resultado: Martes
```

CMONTH(<fFecha>)

Esta función convierte un valor de fecha en un nombre de mes en formato de cadena de caracteres.

✍ Los ejemplos siguientes muestran la forma de utilización de CMONTH():

```
? CMONTH(DATE())         // Resultado: Septiembre
? CMONTH(DATE() + 45)    // Resultado: Octubre
```

CTOD(<cFecha>) --> fFecha

Es una función de conversión de caracteres que convierte una cadena de caracteres en una fecha. Para inicializar una fecha vacía para la introducción de la fecha, especifique <cFecha> como cadena nula (""), SPACE(8) o " / / ". CTOD() se utiliza siempre que necesite un valor de fecha en forma literal.

Argumentos: ***<cFecha>*** es una cadena de caracteres que consiste en números que representan el mes, día y año, separados por un carácter no numérico. Las cifras del mes, día y año debe formatearse con SET DATE.

✍ Este ejemplo utiliza CTOD() para inicializar dos variables de fecha, utilizando una como GET y la otra para la validación RANGE:

```
dInicio := CTOD("01-26-1996")
dActual := CTOD("")
@ 10, 10 SAY "Introduzca la fecha:" GET dActual RANGE dInicio, DATE()
READ
```

DTOC(<fFecha>)

Esta función convierte un valor de fecha en una cadena de caracteres DTOC() devuelve una cadena de caracteres que representa un valor de fecha. El valor de retorno se formatea en el formato de fecha actual. El formato por defecto es mm/dd/aa. Una fecha nula devuelve una cadena de espacios de longitud igual al formato de fecha actual.

Argumentos: ***<fFecha>*** es el valor de fecha que se va a convertir.

✍ Estos ejemplos muestran la utilización general de DTOC():

```
? DATE()                    // Resultado: 09/01/90
? DTOC(DATE())              // Resultado: 09/01/90
? "Hoy es " + DTOC(DATE())  // Resultado: Hoy es 09/01/90
```

DTOS(<fFecha>)

Esta función convierte un valor de fecha en una cadena de caracteres en formato aaaammdd.

Argumentos: ***<fFecha>*** es el valor de fecha que se va a convertir.

✍ Estos ejemplos muestran la forma de utilizar DTOS() con otras funciones:

```
? DATE()                    // Resultado: 09/01/90
? DTOS(DATE())              // Resultado: 19900901
? LEN(DTOS(CTOD("")))       // Resultado: 8
```

✍ El ejemplo muestra cómo crear un índice con una fecha compuesta y una clave de caracteres utilizando DTOS():

```
USE Ventas NEW
INDEX ON DTOS(Fecha) + Vendedor TO NomFecha
```

TIME()

Es una función horaria que muestra la hora del sistema en la pantalla o la imprime en un informe. TIME() devuelve la hora del sistema como una cadena de caracteres en el formato hh:mm:ss. hh es la hora en formato de 24 horas, mm son los minutos y ss son los segundos.

✍ El ejemplo muestra el resultado de utilizar TIME() con SUBSTR() para extraer los dígitos de horas, minutos y segundos:

```
? TIME()                    // Resultado: 10:37:17
? SUBSTR(TIME(), 1, 2)      // Resultado: 10
? SUBSTR(TIME(), 4, 2)      // Resultado: 37
? SUBSTR(TIME(), 7, 2)      // Resultado: 17
```

TONE(<nFrecuencia>, <nDuración>)

Es una función de sonido que sirve para indicar al usuario los diferentes estados de un programa. Pueden ser estados de error, condiciones límite o el final de un proceso muy lento.

Argumentos: ***<nFrecuencia>*** es un valor numérico que indica la frecuencia del tono que va a sonar. ***<nDuración>*** es un número positivo que indica la duración del tono medida en incrementos de 1/18 de segundo. Por ejemplo, una <nDuración> de 18 representa un segundo.

✍ Este ejemplo muestra diferentes tonos:

```
TONE(300, 10)
TONE(100, 8)
TONE(300, 7)
TONE(100, 10)
```

NOTE

Este mandato coloca un comentario de un renglón en un archivo de programa (PRG). NOTE es un mandato de compatibilidad y, por tanto, su utilización no es recomendable. Se sustituye por los símbolos de comentarios doble barra inclinada (//), asterisco (*), doble ampersand (&&) o barra inclinada-asterisco (/*) y asterisco-barra inclinada (*/). Si necesita realizar un comentario de varias líneas, comience el bloque de comentario con un símbolo de barra inclinada y asterisco (/*) y finalícelo con un símbolo de asterisco y barra inclinada (*/).

✍ Estos ejemplos muestran los diferentes símbolos de comentario utilizables en CA-Clipper:

```
SET COLOR TO B+ // El uso de la doble diagonal para un comentario, después de una sentencia de Clipper
/*TONE(300, 1) El uso de la barra inclinada-asterisco  y asterisco-barra inclinada,
TONE(100, 1)     cancela estas 2 líneas de código de Clipper */
SET COLOR TO B+ && Usando doble ampersand para un comentario, después de una sentencia d Clipper
* SET COLOR TO B+ El uso del asterisco para un comentario, cancela el contenido de la línea de Clipper
```

RUN | * <xcLíneaMandato>

RUN ejecuta un programa o mandato DOS desde una aplicación compilada.

Argumentos: ***<xcLíneaMandato>*** es cualquier programa ejecutable y algunos mandatos residentes del DOS. Puede especificarse como cadena literal o como expresión de caracteres encerrada entre paréntesis.

✍ Este ejemplo utiliza RUN para ejecutar un programa (EXE) de la unidad A.

```
RUN A:\PRESENTA.EXE
```

✍ Este ejemplo utiliza RUN para ejecutar un programa (EXE), de la ruta actual (unidad y/o subdirectorio actual):

```
RUN PORTADA
```

QUIT

Este mandato termina la ejecución del programa, cierra los archivos abiertos y regresa el control al sistema operativo.

✍ Este ejemplo utiliza QUIT en un panel de diálogo:

```
IF DialogSiNo(10, 10, "Salir a DOS", "BG+/B,B/W", 2)
   QUIT
ENDIF
RETURN
```

COMANDOS DEL ENTORNO GLOBAL

SET BELL ON/OFF

Este mandato determina si activa la bocina de la computadora durante las operaciones de entrada de datos.

Argumentos: ***ON*** activa la señal acústica ***OFF*** desactiva el sonido.

Si se ha establecido SET BELL ON, la señal suena en las siguientes situaciones:

- El usuario introduce un carácter en la última posición en un GET.

- El usuario intenta escribir datos no válidos en un GET. Los datos son validados por el tipo de datos de la variable GET, la plantilla PICTURE y la cláusula RANGE. La violación de una condición VALID no hace sonar la señal, cualquiera que sea el estado de SET BELL.

SET CURSOR ON/OFF

Este mandato determina si es visible en pantalla el cursor o no.

Argumentos: ***ON*** activa la visualización del cursor. ***OFF*** desactiva la visualización del cursor.

✍ Este ejemplo muestra un uso típico de SET CURSOR:

```
SET CURSOR ON
LOCAL lRespuesta := .F.
@ 24, 15 SAY "¿Salir? [S/N]"  GET lRespuesta PICTURE "S"
READ
SET CURSOR OFF
```

SETCURSOR([<nFormaCursor>])

Es una función de entorno que controla la forma del cursor de pantalla. La forma real depende del controlador actual de la pantalla. SETCURSOR(0) equivale a SET CURSOR OFF y cualquier valor entero positivo de <nFormaCursor> menor que 5 equivale a SET CURSOR ON.

Argumentos: ***<nFormaCursor>*** es un número que indica la forma del cursor. Para códigos sencillos, el fichero de cabecera Setcurs.ch ofrece nombres descriptivos para las diversas formas del cursor tal y como se muestra en la siguiente tabla:

Formas del cursor

Formas	Valor	Setcurs.ch	Bloque medio inferior	2	SC_INSERT
Ninguna	0	SC_NONE	Bloque completo	3	SC_SPECIAL 1
Subrayado	1	SC_NORMAL	Bloque medio superior	4	SC_SPECIAL 2

✍ Este ejemplo utiliza SETCURSOR() para que el cursor adopte la forma de bloque completo para el READ posterior. Una vez finalizado el READ, SETCURSOR() desactiva el cursor:

```
#include "Setcurs.ch"
USE Cliente NEW
@ 10, 10 GET Cliente->Nombre
@ 11, 10 GET Cliente->Telefono
SETCURSOR(3)          // Cambiar el cursor a bloque
READ
SETCURSOR(0)          // Desactivar cursor
```

SET DATE FORMAT [TO] <cFormatoFecha>

SET DATE [TO] AMERICAN | ansi | british | french | german | italian | japan | usa

Este mandato establece el formato de fecha para su introducción y visualización.

Argumentos: ***<cFormatoFecha>*** es una expresión de caracteres que especifica directamente el formato de fecha cuando se especifica la cláusula FORMAT. El día se representa como dd, el mes como mm y el año como yy o yyyy. Cuando no se utiliza la cláusula FORMAT, una de las palabras clave describe el formato de fecha. En la tabla siguiente se muestra el formato para cada valor de palabra clave:

Formatos de SET DATE

Valor	Formato	Valor	Formato	Valor	Formato
AMERICAN	mm/dd/yy	FRENCH	dd/mm/yy	JAPAN	yy/mm/dd
ANSI	yy.mm.dd	GERMAN	dd.mm.yy	USA	mm-dd-yy
BRITISH	dd/mm/yy	ITALIAN	dd-mm-yy		

✍ En este ejemplo la cláusula FORMAT especifica directamente el formato de fecha:

```
SET DATE FORMAT "yyyy:mm:dd"
```

✍ En este ejemplo se utiliza SET DATE para establecer la manera de visualizar la fecha:

```
SET DATE TO BRITIHS  //Resultado:  "dd/mm/yy"
```

SET DEAFULT TO <xCamino>

Este mandato especifica la unidad de disco y directorio por omisión para la creación y grabación de los archivos (DBF).

Argumentos: ***TO <xCamino>*** identifica una unidad de disco y un directorio como vía por defecto y puede indicarse como una especificación de vía literal o como una expresión de caracteres entre paréntesis. Si escribe una unidad y un directorio, debe incluir el carácter ":" después de la letra de la unidad.

✍ Este ejemplo muestra un uso típico de SET DEFAULT:

```
SET DEFAULT TO C:\CLIPPER\FILES
? FILE("Ventas.dbf")        // Resultado: .T.
SET DEFAULT TO C:           // Cambiar unidad por defecto
SET DEFAULT TO \            // Cambiar a directorio raíz
SET DEFAULT TO ..           // Cambiar a directorio padre
```

SET DELETED ON/OFF

Este mandato determina si los registros marcados para su eliminación se procesan o no.

Argumentos: ***ON*** ignora los registros borrados. ***OFF*** procesa los registros borrados.

✍ Este ejemplo ilustra el efecto de la utilización de SET DELETED:

```
USE Ventas NEW
? LASTREC()              // Resultado: 84
DELETE RECORD 4
nRecuento:=RECNO()
? nRecuento              // Resultado: 84
SET DELETED ON
nRecuento:=RECNO()
? nRecuento              // Resultado: 83
```

SET DEVICE TO SCREEN/PRINTER

Este mandato determina si los resultados de un @...SAY se desplegarán en la pantalla o en la impresora. Cuando el dispositivo que se establece es PRINTER, los mandatos @...SAY se envían a la impresora y no se muestran en la pantalla. Además, los mandatos @...SAY se ciñen al valor actual de SET MARGIN.

Cuando se envían los mandatos @...SAY a la impresora, CA-Clipper lleva a cabo un EJECT automático cuando la posición actual de la fila del cabezal de impresión es menor que la última posición de fila de impresión.

Argumentos: ***TO SCREEN*** redirecciona todas las salidas de @...SAY a la pantalla, independientemente de la configuración de SET PRINTER. ***TO PRINTER*** redirecciona la salida de @...SAY al dispositivo establecido con SET PRINTER TO. Esto puede incluir un puerto de impresora local, un spooler de red o un fichero.

✍ Este ejemplo redirecciona la salida de @...SAY a la impresora:

```
SET DEVICE TO PRINTER
@ 2,10 SAY "Hola"
EJECT
```

✍ Este ejemplo dirige el resultado de @...SAY a un fichero:

```
SET PRINTER TO Resultad.txt
SET DEVICE TO PRINTER
@ 10, 10 SAY "Este fichero es: Resultad.txt"
@ 11, 10 SAY DATE()
SET PRINTER TO                  // Cerrar fichero de resultados
SET DEVICE TO SCREEN
```

SET ESCAPE ON/OFF

Este mandato determina si es posible emplear la tecla Esc como tecla de salida durante un READ. Si se ha establecido SET ESCAPE ON, Esc finaliza el READ actual. Los cambios efectuados en el objeto Get actual se pierden y la validación con RANGE o VALID se ignora. Cuando se ha establecido SET ESCAPE OFF y el usuario pulsa Esc, se ignora la pulsación de tecla.

Argumentos: ***ON*** activa Esc como una tecla de salida de READ. ***OFF*** desactiva Esc como una tecla de salida de READ.

✍ Este ejemplo desactiva una posible salida con la tecla Esc de un Read:

```
SET ESCAPE OFF
@ 10, 10 SAY "Nombre: " GET cNombre
READ
```

SET SCOREBOARD ON/OFF

Este mandato activa o desactiva la visualización de mensajes en la línea cero con READ. Cuando se establece SET SCOREBOARD ON, READ muestra mensajes para errores de rango, fechas incorrectas y estado de inserción. Para suprimir la visualización automática de estos mensajes, establezca SET SCOREBOARD OFF.

Argumentos: ***ON*** permite la visualización de mensajes durante un READ o un MEMOEDIT() en la línea cero de la pantalla. ***OFF*** suprime estos mensajes.

✍ Este ejemplo desactiva la visualización de posible mensajes en pantalla como (Insert):

```
SET SCOREBOARD OFF
@ 10, 10 SAY "Nombre: " GET cNombre
READ
```

SETPOS(<nFila>, <nCol>)

Es una función de entorno que desplaza el cursor a una nueva posición en la pantalla.

Argumentos: ***<nFila> y <nCol>*** definen la nueva posición de pantalla del cursor. Estos valores pueden estar comprendidos entre 0,0 y MAXROW(), MAXCOL().

✍ Este ejemplo desplaza el cursor a una nueva posición, después muestra una cadena en la pantalla:

```
SETPOS(1, 1)
?? "Hola a todos"
```

FILE(<cEspecFichero>)

Es una función de entorno que determina si se ha localizado algún fichero que coincida con un patrón de especificación de fichero. FILE() busca el directorio especificado si se ha especificado una vía de forma explícita. Si no se ha especificado una vía, FILE() busca en el directorio por defecto de CA-Clipper y posteriormente, la vía de CA-Clipper. Observe también que FILE() no reconoce los ficheros ocultos o de sistema en su búsqueda.

Argumentos: ***<cEspecFichero>*** determina una especificación de fichero estándar que puede incluir caracteres comodín * y ?, así como una referencia de unidad y de vía. Las referencias explícitas a un fichero deben incluir también una extensión.

✍ En este ejemplo FILE() intenta localizar Ventas.dbf en otro directorio de CA-Clipper, que no sea el utilizado por defecto:

```
? FILE("Ventas.dbf")                  // Resultado: .F.
? FILE("\APPS\DBF\Ventas.dbf")        // Resultado: .T.
SET DEFAULT TO C:\APPS\DBF
? FILE("Ventas.dbf")                  // Resultado: .T.
```

COMANDOS PARA BASES DE DATOS

USE [<xcBase de datos>
[INDEX <xcLista Indices>]

Este mandato abre un fichero existente de base de datos (.dbf), su fichero memo asociado (.dbt) y, opcionalmente, los ficheros de índice asociados (.ntx o .ndx) en el área de trabajo actual o en la siguiente área de trabajo disponible. En CA-Clipper, es posible tener 250 áreas de trabajo y un máximo de 255 ficheros abiertos con el DOS 3.3 y versiones posteriores. Antes de que USE abra un fichero de base de datos y sus ficheros asociados, cierra cualquier fichero activo abierto en el área de trabajo. Cuando se abre un fichero de base de datos, el puntero de registro se sitúa en el 1er. registro lógico del fichero (sino hay fichero de índice especificado.

Argumentos: ***<xcBase de datos>*** es el nombre del fichero de base de datos que se va a abrir. Puede especificarse como un nombre literal o como una expresión de caracteres entre paréntesis. ***INDEX <xcLista Indices>*** especifica los nombres de 1 a 15 ficheros de índice que se van a abrir en el área de trabajo actual.

Nota: Selección del máximo de ficheros abiertos: El control del número de manejadores de fichero disponibles para una aplicación de CA-Clipper depende del mandato FILES del fichero CONFIG.SYS y del parámetro F de la variable de entorno CLIPPER. El parámetro F especifica el número máximo de ficheros que pueden abrirse simultáneamente en un programa de CA-Clipper. CA-Clipper determina el número de ficheros que puede abrir con el menor de estos dos parámetros. Por ejemplo, si el mandato FILES es 120 y

el parámetro F es 50, el número máximo de ficheros que puede abrirse es 50. El límite de ficheros está controlado también por el sistema operativo. En las versiones del DOS anteriores a la 3.3, el número máximo de ficheros que pueden abrirse simultáneamente es 20. En el DOS versión 3.3 y superiores, el límite máximo es 255.

INDEX ON <expClave> TO <xcIndice> [UNIQUE] [<ámbito>] [ASCENDING | DESCENDING]

Es un mandato de base de datos que crea un fichero que contiene un índice de los registros de la base de datos actual, basado en <expClave>. Cuando se utiliza el fichero índice, los registros del fichero de base de datos aparecen en el orden de la expresión clave aunque el índice no altera su orden físico. INDEX ordena las claves de caracteres según el valor ASCII de cada carácter de la cadena, los valores numéricos por orden numérico, las fechas por orden cronológico (las fechas en blanco se tratan como valores inferiores) y los valores lógicos clasificados con los valores verdaderos (.T.) como valores más altos.

Cuando se invoca INDEX ON, se cierran todos los ficheros índice abiertos en el área de trabajo actual y se crea el nuevo fichero índice. Cuando termina la operación de indexado, el nuevo índice permanece abierto, pasando a ser el índice activo y el puntero de registros se coloca en el primer registro del índice.

Argumentos: ***<expClave>*** es una expresión que devuelve el valor clave que se debe colocar en el índice por cada registro del área de trabajo actual. <expClave> puede ser de tipo carácter, fecha, lógico o numérico. La longitud máxima de una expresión clave de índice puede ser de 250 caracteres. ***TO <xcIndice>*** especifica el nombre del fichero índice que va a crearse. Puede especificar el nombre de fichero bien como un nombre literal o como una expresión de caracteres encerrada entre paréntesis. Si el nombre especificado no incluye la extensión del fichero, se utiliza la extensión por defecto. Para índices estándar de CA-Clipper la extensión por defecto es (.ntx). Para índices compatibles con dBASE III PLUS la extensión por defecto es (.ndx). ***UNIQUE*** especifica que <xcIndice> incluye sólo valores clave únicos. ***<ámbito>*** es la parte de la base de datos actual a la que se aplica INDEX. El valor por defecto es todos (ALL) los registros. Se permite cualquier otra cláusula válida de ámbito (p. ej. NEXT <nRegistros>, REST y RECORD <nRegistro>). Si especifica un ámbito, la base de datos se procesa en el orden del índice activo. El ámbito es temporal (es decir, no se guarda en el fichero (.ntx) y no se utiliza para REINDEX).

✍ Este ejemplo crea un índice sencillo:

```
USE Cliente
INDEX ON Nombre TO Cliente
```

✍ Este ejemplo crea un índice con el atributo de unicidad:

```
USE Cliente NEW
INDEX ON Nombre TO Cliente UNIQUE
```

✍ Este ejemplo crea un índice sencillo en orden descendente:

```
USE Cliente NEW
INDEX ON Nombre TO Cliente DESCENDING
```

APPEND BLANK

Es un mandato de base de datos que añade un nuevo registro al final del fichero actual y, seguidamente, lo convierte en el registro activo. Los nuevos campos se inicializan con los valores vacíos de cada tipo de datos.

✍ El ejemplo siguiente añade un registro a un fichero de base de datos:

```
USE Ventas
<sentencias>
APPEND BLANK
```

DELETE [<ámbito>] [WHILE <lCondición>]
[FOR <lCondición>]

Es un mandato de base de datos que marca los registros para su posterior eliminación de la base de datos activa. Estos registros no son eliminados físicamente hasta que se ejecuta la orden PACK y pueden ser recuperados con la orden RECALL antes de su eliminación física.

Argumentos: ***<ámbito>*** es la parte del fichero actual que va a borrarse con DELETE. Si no se especifica un ámbito, DELETE sólo actúa en el registro actual. Si se especifica una cláusula condicional, el valor por defecto es todos los registros. ***WHILE <lCondición>*** especifica el conjunto de registros que cumplen

<lCondición> a partir del registro actual hasta que la condición deje de cumplirse. ***FOR <lCondición>*** especifica el grupo condicional de registros que van a borrarse con DELETE en el ámbito dado.

✍ El ejemplo muestra la utilización de la cláusula FOR para marcar como borrados un conjunto de registros:

```
USE Ventas INDEX Vendedor NEW
DELETE ALL FOR Inactivo
```

GO[TO] <nRegistro> BOTTOM/TOP

Este mandato lleva el apuntador de registros al registro especificado en el área activa o directamente al registro inicial (TOP) o al último registro (BOTTOM). Si hay algún archivo de índices abiertos, el apuntador se mueve de acuerdo al índice en control de la base de datos.

Argumentos: ***<nRegistro>*** especifica el número de registro destino. ***BOTTOM*** especifica el último registro lógico en el área de trabajo actual. ***TOP*** especifica el primer registro lógico en el área de trabajo actual.

✍ Estos ejemplos muestran los resultados de mandatos GO simples:

```
USE Ventas
GO TOP
? RECNO()          // Resultado: 1
GO BOTTOM
? RECNO()          // Resultado: 84
GO 5
? RECNO()          // Resultado: 5
```

PACK

Es un mandato de base de datos que elimina todos los registros marcados para borrar del fichero de base de datos actual, vuelve a indexar todos los índices activos del área de trabajo y recupera todo el espacio físico ocupado por los registros borrados.

✍ A continuación se muestra el resultando de un mandato PACK sencillo:

```
USE Ventas NEW
? LASTREC()            // Resultado: 84
DELETE RECORD 4
PACK
? LASTREC()            // Resultado: 83
```

RECALL [<ámbito>] [WHILE <lCondición>] [FOR <lCondición>]

Es un mandato de base de datos que restablece los registros marcados para borrarse del área de trabajo actual. Es el opuesto del mandato DELETE. Si se ha establecido SET DELETED ON, RECALL puede restablecer el registro actual o un registro específico, si se especifica un ámbito RECORD. Recuerde que una vez que haya purgado un fichero de base de datos, todos los registros marcados se habrán suprimido físicamente del fichero y no podrán recuperarse.

Argumentos: ***<ámbito>*** es la sección del fichero actual de base de datos que se desea recuperar. El ámbito por defecto es el registro actual o NEXT 1. Si se especifica una condición, el ámbito por defecto se convierte en todo el fichero. ***WHILE <lCondición>*** especifica un grupo de registros que cumplen una condición, desde el registro actual hasta que la condición deje de cumplirse. ***FOR <lCondición>*** especifica el grupo condicional de registros que se va a recuperar en el ámbito dado.

✍ Este ejemplo muestra diversos resultados de RECALL:

```
USE Ventas NEW
DELETE RECORD 4
? DELETED()            // Resultado: .T.
RECALL
? DELETED()            // Resultado: .F.
```

REPLACE <idCampo> WITH <exp>
[, <idCampo2> WITH <exp2>...]
[<ámbito>] [WHILE <lCondición>] [FOR <lCondición>]

Es un mandato que asigna nuevos valores a una o más variables de campo del registro actual, en las áreas de trabajo especificadas. Las variables de campo de destino pueden ser de tipo caracteres, fechas, lógicas, memo o numéricas. REPLACE realiza la misma función que el operador de asignación (:=), excepto en que supone que una referencia sin alias se aplica a una variable de campo. Si no se especifica un ámbito o condición, el ámbito por defecto es el registro actual. Si se especifica, la operación de sustitución se realiza en todos los registros que cumplan la condición o que se encuentren en ese ámbito.

Argumentos: ***<idCampo>*** es el nombre de la variable de campo a la que se va a asignar un nuevo valor. Si <idCampo> va precedido de un alias, la asignación tiene lugar en el área de trabajo designada. ***WITH <exp>*** define el valor que se va a asignar a <idCampo>. ***<ámbito>*** es la sección del fichero actual de base de datos que en el que se va a realizar la sustitución. El valor por defecto es el registro actual o NEXT 1. Si se especifica una condición, el valor por defecto es todos los registros del área de trabajo actual. ***WHILE <lCondición>*** especifica un grupo de registros que cumplen una condición, desde el registro actual hasta que la condic.deje de cumplirse. ***FOR <lCondición>*** especifica el grupo condicional de registros que se va a sustituir en el ámbito dado.

✍ Este ejemplo muestra la utilización básica de REPLACE:

```
USE Cliente NEW
APPEND BLANK
USE Facturas NEW
APPEND BLANK
REPLACE Gastos WITH Cliente->Recargo * Coste;
```

SEEK <expBúsqueda>

Es un mandato de base de datos que efectúa una búsqueda en el índice activo. Empieza por la primera clave y continúa el proceso hasta encontrar una coincidencia o existencia de un valor clave superior a <expBúsqueda>. Si existe una coincidencia, el puntero de registro se sitúa en el número de registro encontrado en el índice, en este caso FOUND() devuelve verdadero (.T.). EOF() sólo devuelve verdadero (.T.) si en el índice no hay claves mayores que el argumento de búsqueda.

Argumentos: ***<expBúsqueda>*** es una expresión que debe coincidir con una clave de índice.

✍ Este ejemplo ilustra el mandato SEEK:

```
USE Ventas INDEX Sucursal NEW
SEEK "100"
? FOUND(), EOF(), RECNO()      // Resultado: .T. .F. 1
```

SKIP [<nRegistros>] [ALIAS <idAlias> | <nAreadeTrabajo>]

Este mandato desplaza el puntero de registro a una nueva posición en el área de trabajo activa.

Argumentos: ***<nRegistros>*** es una expresión numérica que determina el número de registros que debe desplazarse el puntero a partir de su posición actual. Un valor positivo desplaza el puntero hacia adelante y uno negativo hacia atrás. ***ALIAS <idAlias>|<nAreadeTrabajo>*** especifica el nombre alias del área de trabajo como una expresión numérica. SKIP sin argumentos desplaza el puntero hacia adelante un registro.

✍ Este ejemplo utiliza SKIP con varios argumentos y muestra el resultado:

```
USE Clientes NEW
SKIP
? RECNO()              // Resultado: 2
SKIP 10
? RECNO()              // Resultado: 12
```

✍ Este ejemplo desplaza el puntero de registro en un área de trabajo no seleccionada actualmente:

```
USE Clientes NEW
USE Facturas NEW
SKIP ALIAS Clientes
```

BROWSE([<nSup>], [<nIzq>], [<nInf>], [<nDer>])

Esta función visualiza registros en una ventana. BROWSE() es una función de interfaz de usuario que invoca, en el área de trabajo actual, un programa de uso general de visualización y edición de registros en formato de tabla. Si desea una lista de las teclas de desplazamiento utilizadas con BROWSE().

Argumentos: <nSup>, <nIzq>, <nInf> y <nDer> definen las coordenadas de las esquinas superior izquierda e inferior derecha de la ventana. Si no se especifican, las coordenadas de ventana por defecto son 1, 0 y MAXROW(), MAXCOL().

Mensajes de Línea de estado de BROWSE

Mensaje	Significado	Mensaje	Significado
<new>	Modo de adición.	<delete>	Borra el registro actual
<bof>	Inicio de fichero.	Registro	Visualiza el número de registro

✍ Este es un ejemplo de visualización de un fichero:

```
USE Fichero1 NEW
BROWSE()
```

EOF()

Esta función determina cuándo se detecta el final del fichero. EOF() devuelve verdadero (.T.) cuando se efectúa un intento de desplazar el puntero de registros sobrepasando el último registro lógico de un fichero de base de datos; de lo contrario, devuelve falso (.F.). Si no hay fichero de base de datos abierto en el área de trabajo actual, EOF() devuelve falso (.F.). Si el fichero de base de datos actual no contiene registros, EOF() devuelve verdadero (.T.).

✍ El ejemplo muestra el uso de EOF() desplazando deliberadamente el puntero del registro más allá del último registro:

```
USE Ventas
GO BOTTOM
? EOF()          // Resultado: .F.
SKIP
? EOF()          // Resultado: .T.
```

FOUND()

Es una función de base de datos que determina si una operación de búsqueda (es decir, FIND, LOCATE, CONTINUE, SEEK o SET RELATION) ha sido satisfactoria. Cuando se ejecuta uno de estos mandatos, FOUND() devuelve verdadero (.T.) si encuentra una coincidencia; de lo contrario, devuelve falso (.F.).

Si el mandato de búsqueda es LOCATE o CONTINUE, una coincidencia es el siguiente registro que cumpla con el ámbito y la condición. Si el mandato de búsqueda es FIND, SEEK o SET RELATION, una coincidencia es la primera clave del índice de control que es igual al argumento de búsqueda. Si el valor clave es igual al argumento de búsqueda, FOUND() devuelve verdadero (.T.); de lo contrario, es falso (.F.). El valor de FOUND() se retiene hasta que se ejecuta otro mandato de movimiento de registro. A menos que sea otro mandato de búsqueda, FOUND() se establece automáticamente en falso (.F.).

✍ Este fragmento de código procesa todos los registros de Cliente con el valor clave "Casanova" utilizando FOUND() para determinar cuándo cambian los valores clave:

```
USE Cliente INDEX Cliente NEW
SEEK "Casanova"
DO WHILE FOUND()
  <sentencias>
   SKIP
   LOCATE REST WHILE Nombre == "Casanova"
ENDDO
```

LASTREC()

Es una función de base de datos que determina el número de registros físicos existentes en el fichero actual de base de datos (.dbf). LASTREC() es idéntica a RECCOUNT(), que se incluye por razones de compatibilidad.

✍ Este ejemplo utiliza una expresión de alias para acceder al número de registros de un fichero de base de datos abierto en un área de trabajo no seleccionada.

```
USE Ventas NEW
USE Cliente NEW
? LASTREC(), Ventas->(LASTREC())
```

SET FILTER TO <lCondición>

Este mandato hace que el archivo de base de datos (DBF) en el área de trabajo activa parezca como si sólo incluyera los registros que cumplen con la condición especificada.

Argumentos: ***TO <lCondición>*** es una expresión lógica que define un conjunto específico de registros del área de trabajo actual accesibles para su proceso.

SET FILTER TO sin argumentos desactiva la condición de filtro.

Nota: Procesa secuencialmente todos los registros del área. Debido a esto, el tiempo necesario para procesar un área de trabajo con filtro será el mismo que el empleado para procesar una sin filtro.

✍ El ejemplo establece un filtro para los registros del fichero Empleado.dbf en los que la edad es superior a 50:

```
USE Empleado INDEX Nombre NEW
SET FILTER TO Edad > 50
LIST Apellido, Nombre, Edad, Telefono
SET FILTER TO
```

COMANDOS PARA EL MANEJO DE VARIAS PANTALLAS

SAVESCREEN([<nSup>], [<nIzq>], [<nInf>], [<nDer>])

Es una función de pantalla que guarda una zona de la pantalla en una variable de cualquier clase de almacenamiento, incluyendo variables de campo. Más adelante, puede volver a mostrar la imagen de pantalla guardada, en la misma posición o en otra nueva, utilizando RESTSCREEN( ). Normalmente, las zonas de pantalla se almacenan y recuperan cuando se utiliza una rutina de menú desplegable o se arrastra un objeto de pantalla.

Argumentos: ***<nSup>, <nIzq>, <nInf>, y <nDer>*** definen las coordenadas de la parte de la pantalla que va a guardarse. Si <nInf> o <nDer> son mayores que MAXROW() o MAXCOL(), la pantalla se recorta. Si no especifican coordenadas, se guarda toda la pantalla (es decir, desde 0,0 hasta MAXROW(), MAXCOL()).

RESTSREEN(<nSuperior>,<nIzquierda>,<nInfeior>,<nDerecha>,<cPantalla>)

Es una función que vuelve a mostrar una zona de pantalla guardada con SAVESCREEN(). La posición de destino puede ser la posición de pantalla original u otra diferente. Si se especifica una nueva posición, la nueva zona debe tener el mismo tamaño que la antigua o se obtendrán resultados ambiguos. Al utilizar RESTSCREEN() para recuperar secciones de pantalla guardadas con SAVESCREEN, pueden especificarse coordenadas comprendidas entre 0, 0,y MAXROW(), MAXCOL().

Argumentos: ***<nSuperior>, <nIzquierda>, <nInferior> y <nDerecha>*** definen las coordenadas de la información de pantalla contenida en ***<cPantalla>***. Si no se especificaron coordenadas con <cPantalla> para que se guardara toda la pantalla, tampoco son necesarias con RESTSCREEN(). ***<cPantalla>*** es una cadena de caracteres que contiene la zona de pantalla guardada.

✍ Este ejemplo guarda una parte de la pantalla y después la restablece:

```
SET COLOR TO BG+/W+
@ 3,35,7,77 BOX "201,205,187,186,188,205,200,186,219”
@ 5,40 SAY "Martín Alejandro Guerra Hdez."
X1:=SAVESCREEN(3,35,7,77)
INKEY(0)
SET COLOR TO B+/W+
@ 0,0,24,79 BOX "201,205,187,186,188,205,200,186,219”
RESTSCREEN(3,35,7,77,X1)          //Observe que las coordenadas sean iguales, a las del Savescreen.
```

COMANDOS PARA IMPRESIÓN DE INFORMES

ISPRINTER()

Es una función de impresora que determina si el puerto paralelo LPT1 está en línea y preparado para imprimir. ISPRINTER() es una función dependiente del hardware, por lo que sólo actúa en sistemas cuyo BIOS es compatible con IBM. Puede comprobar ISPRINTER() para asegurarse de que la impresora está lista

antes de empezar a imprimir. No obstante, si se produce un error durante la impresión, se genera un error de ejecución.

✍ Este ejemplo comprueba si está preparado el puerto paralelo, realizando hasta 25 reintentos. Si el puerto paralelo está preparado, se inicia la operación de impresión:

```
LOCAL nCuenta := 0, nVeces := 25, lPrep
DO WHILE nCuenta++ <= nVeces.AND. !(lPrep := ISPRINTER())
ENDDO
IF lPrep
  REPORT FORM Ventas TO PRINTER
ELSE
  ? "La impresora no está preparada..."
  INKEY(0)
ENDIF
```

EJECT

Este mandato hace avanzar la cabeza de la impresora al inicio de la hoja e inicializa la línea y columna a cero.

✍ En este ejemplo se imprime un informe simple y se utiliza EJECT para avanzar a una nueva página cuando el contador de líneas alcanza el número máximo de líneas de impresión por página:

```
LOCAL nLinea := 99, nPagina := 0
USE Ventas NEW
SET PRINTER ON
DO WHILE !EOF()
  IF nLinea > 55
    EJECT
    ? "Página " + LTRIM(STR(++nPagina, 3))
    ? "Fecha " + CTOD(DATE())
    ? "Vendedor", "Cantidad"
    nLinea := 6
  ENDIF
  ? Ventas->Vendedor, Ventas->Cantidad
  nLinea++
  SKIP
ENDDO
SET PRINTER OFF
CLOSE
```

UNIDAD VII MANEJO DE BASES DE DATOS RELACIONALES

COMANDOS PARA MANEJAR Y RELACIONAR VARIAS BASES DE DATOS.

SELECT <xnAreadeTrabajo> | <idAlias>

Es un mandato de base de datos que cambia las áreas de trabajo. CA-Clipper admite 250 áreas de trabajo, cada una de ellas con un manejador lógico del fichero de base de datos abierto y de sus atributos. Puede referirse a las áreas de trabajo con SELECT bien por su número o por su nombre. El alias de un área de trabajo se asigna automáticamente cuando se utiliza un fichero de base de datos en dicha área o mediante el uso de la cláusula ALIAS.

Argumentos: ***<xnAreadeTrabajo>*** es el número del área de trabajo entre cero y 250 inclusive. Este argumento es una expresión ampliada que puede especificarse como un número literal o como una expresión numérica entre paréntesis. ***<idAlias>*** es el nombre de un área de trabajo existente que hay que seleccionar si existe un fichero de base de datos abierto en ella.

✍ Este ejemplo abre una serie de ficheros de base de datos seleccionando cada área de trabajo por su número y utilizando después cada fichero de base de datos en dicha área de trabajo:

```
SELECT 1
USE Cliente
SELECT 2
```

```
USE Facturas
SELECT 3
USE Referenc
SELECT Cliente
```

✍ Un método mejor es abrir cada base de datos en la siguiente área de trabajo disponible especificando la cláusula NEW en la línea de mandato USE. En este ejemplo, se utiliza USE...NEW en lugar de SELECT y USE:

```
USE Cliente NEW
USE Facturas NEW
SELECT Cliente
```

SET INDEX TO [<xcLista índices>]

Este mandato abre uno o más archivos de índices (NTX) en el área de trabajo activa. Cuando no se especifican los índices se cierran todos los archivos en el área de trabajo.

Argumentos: ***TO <xcLista índices>*** es una lista de hasta 15 nombres de ficheros de índice (.ntx o .ndx) separados mediante comas. Se ignora cualquier referencia a un nombre de fichero de índice que dé como resultado una cadena nula ("") o espacios. Puede especificar cada fichero de índice como un nombre de fichero literal o como una expresión de caracteres entre paréntesis. Si no especifica extensión, el controlador de base de datos actual proporciona una extensión por defecto. (.ntx) es la extensión proporcionada por el controlador de CA-Clipper y (.ndx) la proporcionada por el controlador de dBASE III PLUS.

✍ Este ejemplo muestra la utilización de SET INDEX para abrir varios índices:

```
USE Ventas
SET INDEX TO Iventa
USE CLIENTES
SET INDEX TO Iclient
```

SET RELATION TO [<expCve>/<nReg> INTO <xcAlias>][,[TO]<expCve2>/<nReg2> INTO <xcAlias2>]

Es un mandato de base de datos que enlaza un área de trabajo padre con una o más áreas de trabajo hijas utilizando una expresión clave, número de registro o expresión numérica. Cada área de trabajo padre puede enlazarse con hasta ocho áreas hija. Una relación hace que el puntero de registro se desplace en el área hija de acuerdo con el movimiento del puntero en el área de trabajo padre. Si no se encuentra ninguna coincidencia en el área de trabajo hija, el puntero de registro hijo se sitúa en LASTREC() + 1, EOF() devuelve verdadero (.T.) y FOUND() devuelve falso (.F.).

El método utilizado para enlazar las áreas de trabajo padre e hijas dependen del tipo de <expClave> y de la presencia de un índice activo en el área de trabajo hija. Si ésta tiene un índice activo, la operación de búsqueda es un SEEK estándar. Si no tiene un índice activo y el tipo de <expClave> es numérico, se lleva a cabo una operación GOTO en el área de trabajo hija.

Argumentos: TO ***<expCve>*** es una expresión que lleva a cabo una búsqueda en el área de trabajo hija cada vez que el puntero de registro se desplaza en el área de trabajo padre. Para que este proceso funcione, el área de trabajo hija debe tener un índice en uso. TO ***<nReg>*** es una expresión que lleva a cabo una operación de ir al número de registro coincidente del área de trabajo hija cada vez que el puntero de registro se desplaza en el área de trabajo padre. Si ***<nReg>*** es igual a RECNO(), la relación utiliza el número del registro padre para posicionar el puntero en el mismo número de registro del área de trabajo hija. Para que un tipo de expresión numérica de relación se ejecute correctamente, el área de trabajo hija no debe tener un índice en uso. INTO ***<xcAlias>*** identifica el área de trabajo hija y puede especificarse como un nombre alias literal o como una expresión de caracteres entre paréntesis.

✍ Este ejemplo relaciona tres áreas de trabajo en una configuración múltiple padre-hija con Cliente relacionado con Facturas y Postal:

```
USE Facturas INDEX Facturas NEW
USE Postal INDEX CodPostal NEW
USE Cliente NEW
SET RELATION TO NumClien INTO Facturas, CodPostal INTO Postal
LIST Cliente, Postal->Ciudad, Facturas->Numero, ;
      Facturas->Cantidad
```